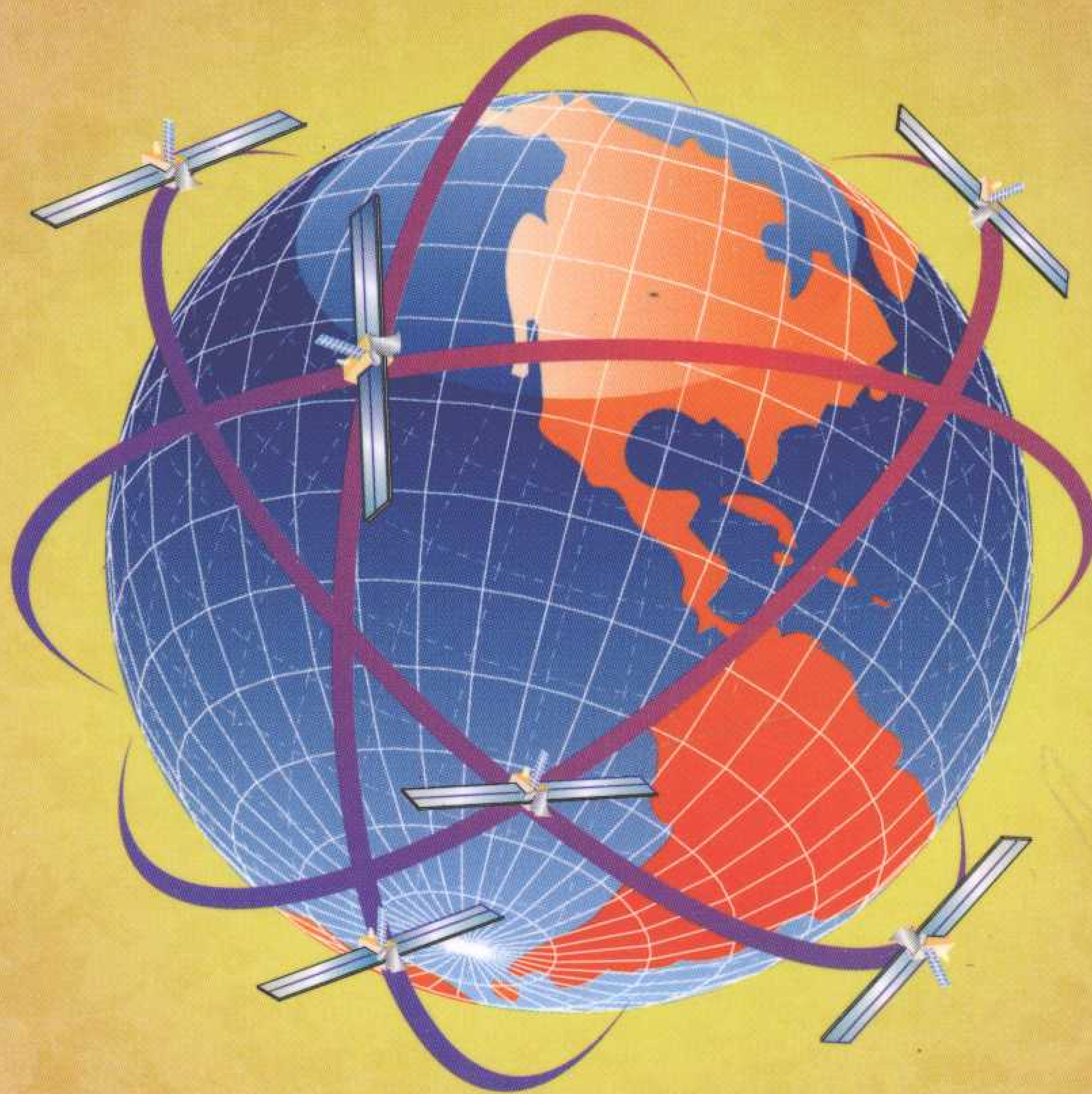
TURING 图灵数学·统计学丛书 41

PEARSON





Numerical Analysis

数值分析

[美] Timothy Sauer 著

吴兆金 王国英 范红军 译



人民邮电出版社
POSTS & TELECOM PRESS

数值分析
Numerical Analysis

"本书结构清晰，条理分明，理论描述精当，实例范围广泛。它突出了数值分析的中心主题，给出了大量的算法及其误差分析，尤其难能可贵的是，它提供了丰富的、取自现实生活各个方面的'实例检验'，显示出作者深厚的理论功底和应用实力。"

——Amazon读者评论

本书是一本优秀的数值分析教材，全面论述了数值分析的基本方法，还介绍了诸如后向误差分析、稀疏矩阵计算及信号处理等新内容。书中实例丰富，涉及计算机、电子、金融等各领域的应用，尤其是专门辟出"实例检验"部分，结合数值分析在各个学科中最新的应用，与MATLAB软件紧密联系，揭示了一种技术或算法可以利用少量的数学知识就能在科技设计中获得巨大的回报。

作者认为，读者不应停留在仅仅学会如何对Newton方法与快速Fourier变换等算法进行编程，还必须吸收那些渗透在数值分析中并把其他相关内容统一起来的伟大思想。收敛性、复杂性、条件作用、压缩以及正交性的概念是这些思想中最重要的，作者通过称为"亮点"的主题格式，强调了现代数值分析中这5个概念的作用。

总之，本书内容生动新颖，实用性强，极富特色，是非常理想的教材和参考书。原书出版不久即被美国多所高校指定为教材或参考书，受到广泛好评。



Timothy Sauer 乔治梅森大学数学系教授。1982年毕业于加州大学伯克利分校，师从著名数学家 Robin Hartshorne。他的主要研究领域为动力系统和数值分析。除本书外，还与人合著有 *CHAOS: An Introduction to Dynamical System* 等书。Sauer 是 *SIAM Journal on Applied Dynamical Systems*、*Journal of Difference Equations and Applications* 和 *Physica D* 等学术期刊的编委。

延伸阅读

Atkinson, Han 《数值分析导论(第3版)》
Demmel 《应用数值线性代数》
Trefethen, Bau 《数值线性代数》
Golub, Van Loan 《矩阵计算(第3版)》
Horn, Johnson 《矩阵分析(英文版)》，卷1和卷2
Lax 《线性代数及其应用(第2版)》
Lay 《线性代数及其应用(第3版•修订版)》
Axler 《线性代数应该这样学》

PEARSON
www.pearsonhighered.com

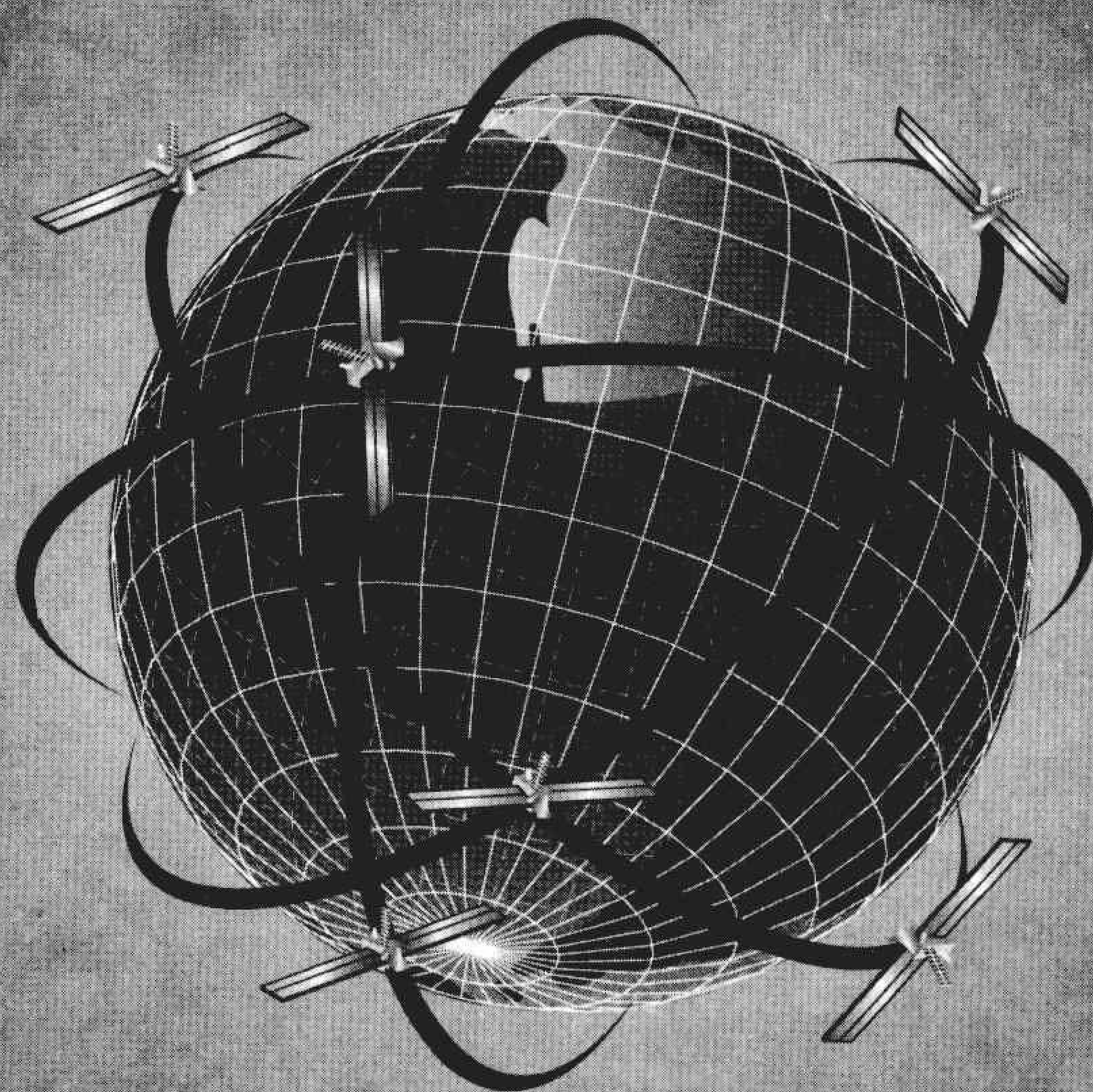
本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010)51095186
反馈/投稿/推荐信箱：contact@turingbook.com
分类建议：数学/计算数学
人民邮电出版社网址 www.ptpress.com.cn

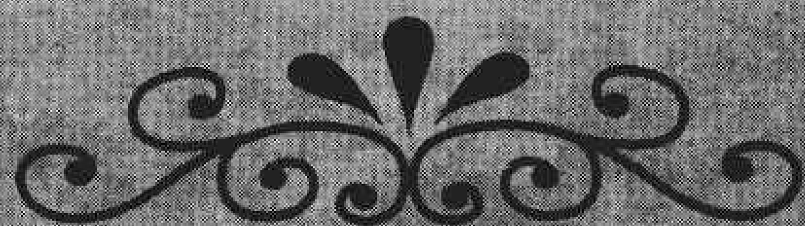




ISBN 978-7-115-21759-2
定价：79.00元

TURING 图灵数学·统计学丛书41





Numerical Analysis

数值分析

[美] Timothy Sauer

吴兆金 王国英 范红军 译

人民邮电出版社

北京

图书在版编目（CIP）数据

数值分析 / (美) 索尔 (Sauer, T.) 著 ; 吴兆金,
王国英, 范红军译. -- 北京 : 人民邮电出版社, 2010.1

(图灵数学·统计学丛书)
书名原文: Numerical Analysis
ISBN 978-7-115-21759-2

Ⅰ. ①数… Ⅱ. ①索… ②吴… ③王… ④范… Ⅲ.
①数值计算 Ⅳ. ①O241

中国版本图书馆CIP数据核字(2009)第206594号

内容提要

本书以收敛性、复杂性、条件作用、压缩和正交性这5个主要思想为核心进行展开. 内容包括求解方程组、插值、最小二乘、数值微分、数值积分、微分方程及边值问题、随机数及其应用、三角插值、压缩、最优化等. 每章都有一个实例检验, 有助于读者了解到相关应用领域. 附录中介绍了矩阵代数和 MATLAB, 并提供了部分习题的答案.

本书内容广泛, 实例丰富, 可作为自然科学、工程技术、计算机科学、数学、金融等专业人员进行教学和研究的参考书.

图灵数学·统计学丛书

数值分析

- ◆ 著 [美] Timothy Sauer
  译 吴兆金 王国英 范红军
  责任编辑 明永玲
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
  邮编 100061 电子函件 315@ptpress.com.cn
  网址 http://www.ptpress.com.cn
  北京铭成印刷有限公司印刷
- ◆ 开本：700×1000 1/16
  印张：36.5
  字数：736千字 2010年1月第1版
  印数：1–3 000册 2010年1月北京第1次印刷

著作权合同登记号 图字： 01-2006-5779号

ISBN 978-7-115-21759-2

定价：79.00元

读者服务热线：(010)51095186 印装质量热线：(010) 67129223

反盗版热线：(010)67171154

版 权 声 明

译 者 序

计算机是 20 世纪以来对人类社会影响最为深刻的高新科技成果之一, 而科学计算已成为当今科学研究中与理论分析和实验研究并列的三种基本手段之一. 科学计算是数学与计算机的有机结合, 而且它本身也成为数学科学自身发展的源泉和途径之一. 数值分析及其有关内容在培养学生科学计算能力上具有不可替代的作用, 目前许多高校已将数值分析列入自然科学、工程技术乃至社会科学的教学计划中.

由于数值分析涉及范围甚广, 应用于诸多的领域, 为使学习者打下较为坚实的理论基础, 了解和使用相关数学软件, 并在此基础上设计和编写自己的算法及程序以解决各类实际应用问题, 选择一本好的教材自然是重要的. 国外最新出版的由乔治・梅森大学 Timothy Sauer 教授编著的这本《数值分析》极富特色, 出版不久即广受好评.

该书内容涵盖非常全面, 其中关于边界值、随机数值分析、三角插值、压缩、特征值、优化等方面的内容是其他教材不多见的. 此外, 作者把许多看似关联不大的技术融合在一本书中, 并强调: 不仅仅要学会如何使用 Newton、Runge-Kutta、快速 Fourier 变换等方法, 还要吸收那些渗透在数值分析中并将各种不同的方法统一起来的主要思想.

本书集中讨论了收敛性、复杂性、条件作用、压缩和正交性的概念及其作用, 并以此分析、评论其他相关的论题, 作者以这 5 个最重要的概念为框架展开内容, 不拘泥于形式的讨论, 向读者传达了什么是数值分析理论中真正至关重要的主题.

作者还介绍了诸如后向误差分析、稀疏矩阵计算及信号处理等概念, 融合了计算机、电子、金融等各领域最新的应用, 给出了大量的实例和图片, 内容生动新颖, 实用性强. 实例检验提供了数值分析在各个学科中最新的应用, 与 MATLAB 的紧密联系可容易地实现制图, 并有效地解决工业规模化问题.

本书结构清晰, 条理分明, 理论描述精当, 实例范围广泛. 除经典问题外, 还涉及许多最新的前沿课题. 每个章节还提供了大量的概念性及计算性的练习 (习题与计算机问题), 帮助读者理解、消化、复习和巩固所学知识, 并可使读者在学会解决各类问题方面逐步积累起经验. 适合于大学生、研究生以及相关人员学习和参考.

本书由吴兆金、王国英和范红军翻译, 其中第 0~1 章、第 3~4 章、第 10 章及索引由吴兆金翻译, 第 5~8 章、第 12 章、习题选解及附录 A 由王国英翻译, 第 2 章、第 9 章、第 11 章、第 13 章及附录 B 由范红军翻译, 全书由吴兆金统稿. 本书

的翻译工作得到人民邮电出版社图灵公司领导和编辑的支持和帮助, 对此向他们表示衷心的感谢.

在本书的翻译过程中, 我们力求忠实、准确地反映原著的内容和风格. 鉴于我们水平所限, 翻译错误及不妥之处在所难免, 恳请读者批评、指正.

译　者

2008 年 3 月于南京大学

译者简介

吴兆金　男, 数学编辑, 副编审. 1978 年毕业于南京大学数学系计算数学专业, 留校任教. 曾任《高等学校计算数学学报》编辑, 现为《Analysis in Theory and Applications》和《南京大学学报数学半年刊》编辑. 已参与出版教材 1 部, 发表学术论文数篇.

王国英　男, 1944 年 11 月生, 1967 年南京大学数学系毕业, 1982 年在南京大学数学系获硕士学位. 现为南京大学数学系信息与计算科学专业的教授, 研究方向是偏微分方程数值解法和奇异摄动问题的数值方法. 曾在核心一级刊物发表研究论文 20 余篇, 已出版译著《数值分析》(全美经典) 和教材《工程数学》.

范红军　男, 1966 年 9 月出生, 1989 年毕业于南京大学数学系并留校任教至今, 现为副教授. 曾编著出版《高等数学》和《大学数学典型题解析》等. 现主要从事数值代数方面的研究.

前　　言

本书是为工程、科技、数学和计算机科学等专业的学生而写的入门教科书, 其目的十分明确: 描述解决科技和工程问题的算法以及讨论算法所需的数学基础, 期望适用于具有初等微积分和矩阵代数基础的学生的主修课程.

作为一门学科, 数值分析的内容极为丰富, 饱含实用思路, 要把很多灵巧但又关联不大的技术用一本书来概括是非常具有挑战性的. 要深入理解, 读者不仅必须学会如何对 Newton 方法、Runge-Kutta 方法与快速 Fourier 变换进行编程, 而且必须吸收那些渗透在数值分析中、把其他相关内容统一起来的伟大思想.

收敛性、复杂性、适用条件、压缩以及正交性的概念是这些思想中最重要的. 任何合适的逼近方法都必须收敛到正确的答案, 尤其是有更多计算资源提供给它时更当如此, 并且计算方法的复杂性也是由资源利用来衡量的. 一个问题的适用条件, 或者对误差放大率的敏感程度, 是了解如何求解问题的基础. 在数值分析的许多最新应用中, 目标是用更短或更浓缩的方式来表示数据. 最后, 正交性在若干领域中对效率的影响是决定性的, 并且在要考虑适用条件或者以压缩性为目标时, 它是不可替代的.

通过称为 "亮点"(Spotlight) 的主题元素, 我们强调了现代数值分析中这 5 个概念的作用. 它们评论当前的论题, 并且联系到书中其他地方出现的相同概念的其他描述. 同时, 我们希望用这种明显的方式突出这 5 个概念, 能够强调当前页面的重点知识, 起到点题之功效.

虽然公认数值分析的思想对现代科技与工程的实践来说是必需的, 但仍需不断强化这一理念. "实例检验" 就给出一些用数值分析方法解决科技问题的具体例子. 这些扩充的应用应时而选并贴近日常的经验. 虽然实例检验不可能 (甚至是不要求) 表现问题的全部细节, 但它试图从一定深度去揭示一种技术或算法可以利用少量的数学知识就在科技的设计中获得巨大的回报.

在本书中, MATLAB 既用于算法说明又用于学生作业和课题的建议平台. 本书中 MATLAB 代码的数量是经过仔细调控的, 这是因为太多的代码会适得其反. 前几章的 MATLAB 代码多一些, 可以使读者逐步熟悉程序. 当提供更加详细的代码时 (比如说, 在插值、常微分和偏微分方程的学习中), 期望读者以此作为探索和拓展的起点.

尽管本书不是非要用到一种计算平台不可, 但 MATLAB 在工程和科技部门的

广泛应用表明, 一种通用的计算机语言可以让很多工作更加顺利. 借用 MATLAB, 所有的界面问题 (数据的输入/输出与绘图等) 都能被一举解决. 数据结构问题 (比如, 研究稀疏矩阵方法时产生的问题) 可以利用适当的命令进行标准化. MATLAB 有专门设施来处理声音和图像文件的输入和输出. 微分方程的模拟是容易实现的, 因为 MATLAB 中有相应的动画命令. 这些目标虽说也可以用其他方式去实现, 但是拥有一个可以运行在几乎所有的操作系统上并简化了细节的软件包, 以便学生能集中精力处理实际的数学问题, 何乐而不为呢? 附录 B 是简短的 MATLAB 指南, 它可以用作学生的 MATLAB 入门介绍或作为已经熟悉该软件的读者的一个参考.

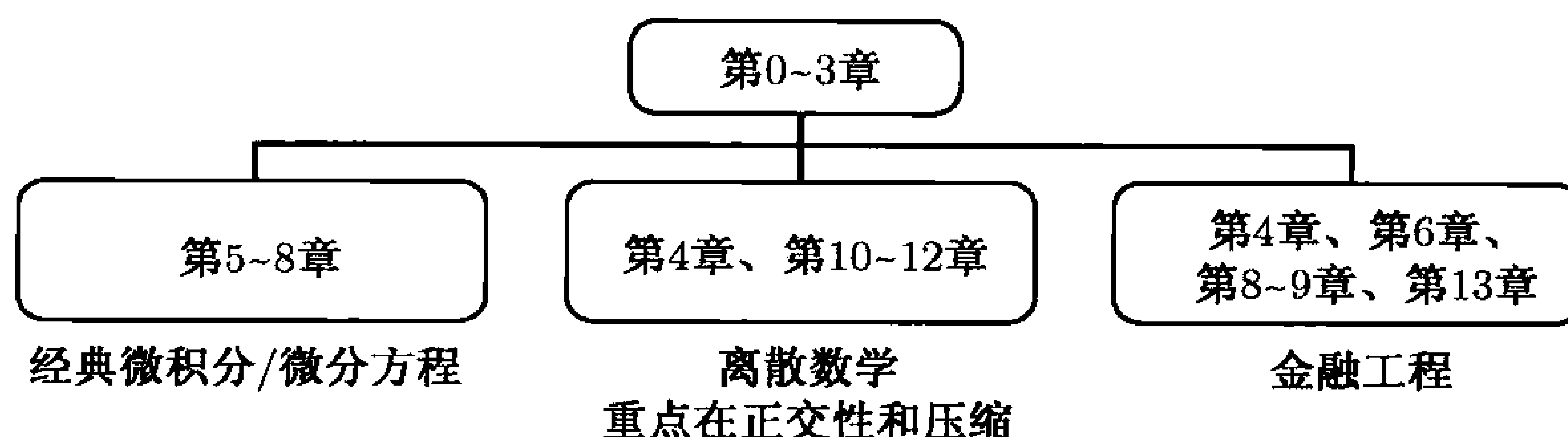
本书附 CD[①]一张, 内容包括直接取自书中的 MATLAB 程序. 这些程序在网站 www.aw-bc.com/sauer 上也可以找到, 该站点上还有新的资源和更新供使用者下载.

本书的结构是先介绍基本思想, 然后描述更复杂的概念. 第 0 章给出了后面要用到的基本知识. 一些教师喜欢从头讲起; 另一些教师 (包括作者本人) 喜欢从第 1 章开始并在需要时折回到第 0 章的一些主题. 第 1 章与第 2 章给出了解方程的各种形式. 第 3 章讲用插值法处理数据拟合, 第 4 章介绍最小二乘法拟合. 在接下来的第 5~8 章, 我们回到连续数学中的经典数值分析领域: 数值微积分, 带初值与边值条件的常微分方程与偏微分方程的数值解.

第 9 章研究随机数, 以便为第 5~8 章提供补充的方法: 可替换标准数值积分格式的蒙特卡罗法以及随机微分方程, 当模型中出现不确定性时这些方法是必要的.

压缩是数值分析的核心主题, 尽管它经常隐藏在插值、最小二乘与 Fourier 分析中. 第 10 章和第 11 章特别讲述了现代压缩技术. 在第 10 章中, 快速 Fourier 变换是作为进行三角插值的一种工具, 包括精确插值和最小二乘近似. 第 11 章介绍离散余弦变换与 Hoffman 编码, 重点并充分讨论了音频压缩的诸多联系, 这两种方法也是现代音频与图像压缩的标准工具. 第 12 章讲述特征值与奇异值, 也强调它们与数据压缩之间的联系, 这在当前的应用中变得越来越重要. 第 13 章是关于优化技术的简短介绍.

如果审慎地选讲一些主题, 本书也可用于一学期的课程. 第 0~3 章是必备的基础. 作为一学期授课的话, 可做如下设计:



① 中文版将原英文书所附 CD 的内容放在了图灵网站 (www.turingbook.com) 上供读者免费注册下载. ——编者注

特别要感谢在撰写本书过程中给予我帮助的许多人, 包括使用本书前几个版本并提出有用建议的学生们. 感谢以下人员, 他们让我避免了一些难堪的错误: Frank Purcell, Paul Lorczak, Steve Whalen, Diana Watson, Joan Saniuk, Robert Sachs, David Walnut, Stephen Saperstone, Tom Wegleitner, Tjalling Ypma. 感谢友好、机智的 Addison Wesley 员工, 包括 William Hoffman, Joanne Ha, Peggy McMahon, Joe Vetere, Emily Portwood, Barbara Atkinson, Beth Anderson 以及在 Westwords 的 Melena Fenn, 正是他们使得本书得以出版. 另外, 还要感谢来自其他大学的下面这些读者, 他们给予我鼓励, 并对改进早期版本提出了宝贵的建议.

Eugene Allgower, 科罗拉多州立大学
Jerry Bona, 伊利诺伊大学芝加哥分校
George Davis, 佐治亚州立大学
Alberto Delgado, 布莱德利大学
Robert Dillon, 华盛顿州立大学
Gregory Goeckel, 长老会学院
Herman Gollwitzer, 德雷塞尔大学
Don Hardcastle, 贝勒大学
David R. Hill, 坦普尔大学
Daniel Kaplan, 麦卡利斯特学院
Lucia M. Kimball, 本特利学院
Seppo Korpela, 俄亥俄州立大学
William Layton, 匹兹堡大学
Doron Levy, 斯坦福大学
Shankar Mahalingam, 加州大学河滨分校
Amnon Meir, 奥本大学
Peter Monk, 特拉华大学
Joseph E. Pasciak, 得克萨斯 A&M 大学
Steven Pav, 加州大学圣地亚哥分校
Jacek Polewczak, 加州州立大学
Jorge Rebaza, 西南密苏里州立大学
Jeffrey Scroggs, 北卡罗来纳州立大学
Sergei Suslov, 亚利桑那州立大学
Daniel Szyld, 坦普尔大学
Ahlam Tannouri, 摩根州立大学
Bruno Welfert, 亚利桑那州立大学

T.S.

目　录

第0章　基　　础

本书的主要目的是介绍和讨论用计算机解数学问题的方法. 最基本的算术运算是加法和乘法. 它们也是计算多项式 $P(x)$ 在特定点 x 处的值所必需的运算. 多项式是我们将要构造的许多计算技术的基石, 这并非巧合.

因此, 了解如何计算多项式是很重要的. 读者可能已经知道如何计算多项式, 并且可能会认为对这种容易的问题花时间近乎荒谬. 但是越是基本的运算, 我们越要把它做得正确. 因此, 我们将考虑如何尽可能高效地进行多项式计算.

0.1　多项式计算

计算在 $x=\frac{1}{2}$ 处的多项式 $P(x)=2x^4+3x^3-3x^2+5x-1$, 最好的方法是什么? 假设多项式的系数和数 $\frac{1}{2}$ 存储在存储器里, 设法使求 $P(\frac{1}{2})$ 所需要的加法和乘法的次数最少. 为了简单, 我们将不考虑把数从存储器存入和取出所花的时间.

方法 1　首先最直接的方法是

$$P\left(\frac{1}{2}\right)=2*\frac{1}{2}*\frac{1}{2}*\frac{1}{2}*\frac{1}{2}+3*\frac{1}{2}*\frac{1}{2}*\frac{1}{2}-3*\frac{1}{2}*\frac{1}{2}+5*\frac{1}{2}-1=\frac{5}{4}. \tag{0.1}$$

它需要 10 次乘法和 4 次加法. 其中两次加法实际上是减法, 但是, 因为减法可以看作是加上一个负的存储数, 所以我们不必担心这种区别.

当然还有比 (0.1) 更好的方法. 工作量是重复的, 通过消除对 1/2 的重复相乘, 可以减少一些运算. 更好的策略是: 首先计算 $(\frac{1}{2})^4$, 同时存储计算过程中的部分积. 这样就导出了以下方法.

方法 2　首先求出输入数 $x=\frac{1}{2}$ 的各次幂, 并把它们存储起来备用:

$$\frac{1}{2}*\frac{1}{2}=\left(\frac{1}{2}\right)^2,\quad \left(\frac{1}{2}\right)^2*\frac{1}{2}=\left(\frac{1}{2}\right)^3,\quad \left(\frac{1}{2}\right)^3*\frac{1}{2}=\left(\frac{1}{2}\right)^4.$$

现在我们就可以把这些项加起来:

$$P\left(\frac{1}{2}\right)=2*\left(\frac{1}{2}\right)^4+3*\left(\frac{1}{2}\right)^3-3*\left(\frac{1}{2}\right)^2+5*\frac{1}{2}-1=\frac{5}{4}.$$

上式中有 3 个 $\frac{1}{2}$ 的乘积以及 4 个相关乘积. 总计我们已减少至 7 次乘法以及 4 次加法. 将运算次数从 14 次减少到 11 次是否是重大的改进? 如果仅是要做一次计

算, 则答案也许是否定的. 不论使用方法 1 或者方法 2, 在你的手指离开计算机键盘之前就可得到答案. 然而, 假如每秒钟需要多项式对不同的输入 x 进行几次计算, 那么能否及时地得到信息, 其间的差别就可能是关键的.

对于 4 次多项式来说, 第 2 种方法是否做得最好呢? 可能很难想象我们还能再减少 3 次运算, 但是我们的确能够做到. 最好的初等方法如下.

方法 3(嵌套乘法) 把多项式改写为下面的形式以便能依括号从内到外进行计算:

$$\begin{aligned} P(x) &= -1 + x(5 - 3x + 3x^2 + 2x^3) = -1 + x(5 + x(-3 + 3x + 2x^2)) \\ &= -1 + x(5 + x(-3 + x(3 + 2x))) = -1 + x * (5 + x * (-3 + x * (3 + x * 2))), \end{aligned} \tag{0.2}$$

这里的多项式是倒过来写的, 而且 x 的乘幂是作为多项式余下部分的因子. 一旦你能看懂这种写法, 就会明白这种改写并不需要计算——系数没有改变. 现在我们从里往外进行计算:

$$\begin{array}{llll} 乘 \ \frac{1}{2} * 2, & 加 \ +3 \to 4; & 乘 \ \frac{1}{2} * 4, & 加 \ -3 \to -1; \\ 乘 \ \frac{1}{2} * (-1), & 加 \ +5 \to \frac{9}{2}; & 乘 \ \frac{1}{2} * \frac{9}{2}, & 加 \ -1 \to \frac{5}{4}. \end{array} \tag{0.3}$$

这种方法称为**嵌套乘法**(nested multiplication) 或 **Horner方法**. 计算该多项式仅用了 4 次乘法和 4 次加法. 通常一个 d 次多项式能用 d 次乘法和 d 次加法进行计算. 嵌套乘法与多项式运算的综合除法密切相关.

这个多项式计算的例子具有科学计算中计算方法所研究的所有问题的特征. 首先, 计算机能够十分迅速地进行非常简单的工作. 其次, 重要的是即使对简单的任务也要尽可能提高工作效率, 因为它们可能要执行许多次. 第三, 最好的方法可能不是显而易见的. 在过去半个多世纪里, 数值分析和科学计算领域与计算机硬件技术密切相关, 对于一些常规的问题已经开发出有效的解题技术.

虽然多项式 $c_1 + c_2x + c_3x^2 + c_4x^3 + c_5x^4$ 的标准形式能写成

$$c_1 + x(c_2 + x(c_3 + x(c_4 + x(c_5)))) \tag{0.4}$$

这种嵌套形式, 但是某些应用要求更一般的形式. 特别地, 第 3 章里的插值计算将需要

$$c_1 + (x - r_1)(c_2 + (x - r_2)(c_3 + (x - r_3)(c_4 + (x - r_4)(c_5)))) \tag{0.5}$$

这种形式, 这里 r_1, r_2, r_3, r_4 称为**基点**(base point). 注意, 在式 (0.5) 中取 $r_1 = r_2 = r_3 = r_4 = 0$ 便恢复到原来的嵌套形式 (0.4).

以下 MATLAB 代码提供了嵌套乘法的一般形式 (与式 (0.3) 比较):

```
%Program 0.1 Nested multiplication
%Evaluates polynomial from nested form using Horner's Method
%Input: degree d of polynomial,
%       array of d+1 coefficients c (constant term first),
%       x-coordinate x at which to evaluate, and
%       array of d base points b, if needed
%Output: value y of polynomial at x
function y=nest(d,c,x,b) if nargin<4, b=zeros(d,1); end y=c(d+1);
for i=d:-1:1
  y = y.*(x-b(i))+c(i);
end
```

运行这个 MATLAB 函数只是置换包括次数、系数、求值点及基点等输入数据. 例如可以用 MATLAB 命令

>> **nest**(4,[−1 5 −3 3 2],1/2,[0 0 0 0])

ans=

1.2500

来计算多项式 (0.2) 在 $x = \frac{1}{2}$ 处的值, 就像我们以前用手算求得的一样. 在执行指令时必须经由 MATLAB 路径 (或在当前目录中) 使用文件`nest.m`. 本书中给出的其余 MATLAB 代码的使用方法与此相同.

若`nest`指令用于如 (0.2) 中所有基点为 0 的情形, 那么使用其简化形式

>> **nest**(4,[−1 5 −3 3 2],1/2)

可以得到同样的结果. 这是由于`nest.m`中的`nargin`语句. 假如输入参数的数量少于 4, 那么就自动将基点设为 0.

由于 MATLAB 中向量记法的无缝处理, 这种`nest`指令可以立即对 x 的一组数值进行计算. 以下代码便可说明这一点:

>> **nest**(4,[−1 5 −3 3 2],[−2 −1 0 1 2])

ans=

−15 −10 −1 6 53

最后, 第 3 章中的 3 次插值多项式

$$P(x) = 1 + x\left(\frac{1}{2} + (x-2)\left(\frac{1}{2} + (x-3)\left(-\frac{1}{2}\right)\right)\right)$$

有基点 $r_1 = 0, r_2 = 2, r_3 = 3$, 可以通过以下代码计算出它在 $x = 1$ 处的值:

>>**n est**(3[1,1/2 1/2 −1/2],1,[0 2 3])

ans=

0

例 0.1 找出一种高效的方法来计算多项式

$$P(x) = 4x^5 + 7x^8 - 3x^{11} + 2x^{14}.$$

改写多项式可以帮助减少所需要的计算次数. 一种想法是从各项中提出因子 x^5, 并把其余部分写成 x^3 的多项式

$$P(x) = x^5(4 + 7x^3 - 3x^6 + 2x^9) = x^5 * (4 + x^3 * (7 + x^3 * (-3 + x^3 * (2)))).$$

首先, 对每一个输入 x, 我们需要计算 $x * x = x^2, x * x^2 = x^3$ 以及 $x^2 * x^3 = x^5$. 这 3 次乘法连同与 x^5 的乘法, 再加上关于 x^3 的 3 次多项式的 3 次乘法和 3 次加法, 就给出了: 每次计算总共需要 7 次乘法和 3 次加法运算. ◀

习题 0.1

1. 改写下列多项式为嵌套形式. 在 $x = \frac{1}{3}$ 时, 分别用嵌套形式和不用嵌套形式进行计算:
   (a) $P(x) = 6x^4 + x^3 + 5x^2 + x + 1$;
   (b) $P(x) = -3x^4 + 4x^3 + 5x^2 - 5x + 1$;
   (c) $P(x) = 2x^4 + x^3 - x^2 + 1$.
2. 改写下列多项式为嵌套形式, 并在 $x = -\frac{1}{2}$ 时计算:
   (a) $P(x) = 6x^3 - 2x^2 - 3x + 7$;
   (b) $P(x) = 8x^5 - x^4 - 3x^3 + x^2 - 3x + 1$;
   (c) $P(x) = 4x^6 - 2x^4 - 2x + 4$.
3. 把 $P(x)$ 看成 x^2 的多项式, 并用嵌套乘法计算当 $x = \frac{1}{2}$ 时 $P(x) = x^6 - 4x^4 + 2x^2 + 1$ 的值.
4. 计算带基点的嵌套多项式 $P(x) = 1 + x\left(\frac{1}{2} + (x-2)\left(\frac{1}{2} + (x-3)\left(-\frac{1}{2}\right)\right)\right)$ 在 (a)$x = 5$ 及 (b)$x = -1$ 处的值.
5. 计算带基点的嵌套多项式 $P(x) = 4 + x(4 + (x-1)(1 + (x-2)(3 + (x-3)(2))))$ 在 (a)$a = \frac{1}{2}$ 及 (b)$x = -\frac{1}{2}$ 处的值.
6. 说明在给定的输入 x 处, 如何用尽可能少的运算计算多项式. 需要多少次乘法和加法?
   (a) $P(x) = a_0 + a_5x^5 + a_{10}x^{10} + a_{15}x^{15}$;
   (b) $P(x) = a_7x^7 + a_{12}x^{12} + a_{17}x^{17} + a_{22}x^{22} + a_{27}x^{27}$.
7. 用一般的嵌套乘法算法, 计算带基点的 n 次多项式需要多少次加法和乘法?

计算机问题 0.1

1. 用函数`nest`计算 $P(x) = 1 + x + \cdots + x^{50}$ 在 $x = 1.000\,01$ 处的值. (用 MATLAB 的`ones`命令省去录入.) 通过与等价表达式 $Q(x) = (x^{51} - 1)/(x - 1)$ 比较, 求计算误差.
2. 用`nest.m`计算 $P(x) = 1 - x + x^2 - x^3 + \cdots + x^{98} - x^{99}$ 在 $x = 1.000\,01$ 处的值. 找出一个更简单的等价表达式, 并用它来估计嵌套乘法的误差.

0.2 二进制数

为了在 0.3 节中深入研究计算机算术, 我们需要了解二进制数系. 为了在计算机上存储数并且简化如加法和乘法这样的计算机运算, 我们把十进制数从以 10 为

基转化到以 2 为基. 把上述过程倒过来就给出十进制表示的输出. 本节讨论十进制数和二进制数之间的转换方法.

二进制数表示如下:

$$\cdots b_2 b_1 b_0 . b_{-1} b_{-2} \cdots,$$

这里每一个二进制数字或者每一**位**(bit) 数是 0 或者 1. 把这个数写成以 10 为基的形式就是

$$\cdots b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} \cdots,$$

例如, 十进制数 4 表示为以 2 为基的形式是 $(100.)_2$, 而 $\frac{3}{4}$ 可表示为 $(0.11)_2$.

0.2.1 十进制到二进制的转换

我们把十进制数 53 表示为 $(53)_{10}$, 以强调它是以 10 为基的. 为转换成二进制, 最简单的方法是把这个数分成整数和小数两部分, 再分别转换. 例如数 $(53.7)_{10} = (53)_{10} + (0.7)_{10}$. 我们把每一部分转换为二进制后再把结果合并起来.

整数部分 把十进制整数转换为二进制的方法是: 逐次与 2 相除并记录余数. 从小数点开始记录余数 0 或 1, 并自右向左移动余数. 例如对于 $(53)_{10}$, 我们有

$$\begin{array}{lll} 53 \div 2 = 26 \quad 余\ 1, & 26 \div 2 = 13 \quad 余\ 0, & 13 \div 2 = 6 \quad 余\ 1, \\ 6 \div 2 = 3 \quad 余\ 0, & 3 \div 2 = 1 \quad 余\ 1, & 1 \div 2 = 0 \quad 余\ 1, \end{array}$$

因此, 以 10 为基的数 53 能表成二进制数 110101, 即记为 $(53)_{10} = (110101.)_2$. 检查这个结果, 我们有 $110101 = 2^5 + 2^4 + 2^2 + 2^0 = 32 + 16 + 4 + 1 = 53$.

小数部分 把上面各步反过来就能把 $(0.7)_{10}$ 转换成二进制数, 逐次用 2 相乘并记录整数部分, 然后从小数点向右移动记录的整数:

$$\begin{array}{lll} 0.7 \times 2 = 0.4 + 1, & 0.4 \times 2 = 0.8 + 0, & 0.8 \times 2 = 0.6 + 1, \\ 0.6 \times 2 = 0.2 + 1, & 0.2 \times 2 = 0.4 + 0, & 0.4 \times 2 = 0.8 + 0, \cdots \end{array}$$

注意, 此过程每 4 步重复一次, 并且以完全相同的方式无穷多次重复. 因此,

$$(0.7)_{10} = (0.1011001100110 \cdots)_2 = (0.1\overline{0110})_2,$$

其中上面一杠的记号表示无穷多次重复的数位. 把这两部分结合起来, 我们得到

$$(53.7)_{10} = (110101.1\overline{0110})_2.$$

0.2.2 二进制到十进制的转换

把二进制数转换成十进制, 最好也分成整数和小数两部分.

整数部分 如以前做的那样, 只要把 2 的各次幂简单地相加即可. 二进制数 $(10101)_2$ 可简单地写成 $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (21)_{10}$.

小数部分 如果小数部分是有限的 (以 2 为基的展开式是有限的), 可用同样的方法进行. 例如,

$$(0.1011)_2 = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = \left(\frac{11}{16}\right)_{10}.$$

只有当小数部分不是有限的以 2 为基的展开式时, 问题才变得复杂起来. 有几种方法能把无穷多次重复的二进制展开式转换成十进制分数. 利用乘 2 以后便移位的性质或许是最简单的方法.

例如, 假设要把 $x = (0.\overline{1011})_2$ 转换为十进制, 把 x 乘以 2^4, 并把它放在二进制数的左边, 然后再减去原来的数 x:

$$\begin{aligned} 2^4 x &= 1011.\overline{1011} \\ x &= 0000.\overline{1011} \end{aligned}$$

相减后得到

$$(2^4 - 1)x = (1011)_2 = (11)_{10}$$

解出 x, 于是得到以 10 为基的数 $x = \frac{11}{15}$.

作为另一个例子, 假定小数部分并不是一开始就重复, 例如 $x = 0.10\overline{101}$, 乘以 2^2, 移位后得到 $y = 2^2 x = 10.\overline{101}$. y 的小数部分, 称之为 $z = 0.\overline{101}$, 像以前一样计算:

$$\begin{aligned} 2^3 z &= 101.\overline{101} \\ z &= 000.\overline{101}. \end{aligned}$$

因此 $7z = 5$, 所以 $y = 2 + \frac{5}{7}, x = 2^{-2} y = \frac{19}{28}$, 它是以 10 为基的. 把 $\frac{19}{28}$ 转换成二进制数, 并与原来的 x 进行比较来检查这个结果, 不失为一个很好的练习.

二进制数是机器计算的基础. 但是人们认为它们太长而不便于理解. 有时会用基 16, 只是为了更容易地表示一个数. **十六进制数**(hexadecimal number) 是用 16 个数字 $0, 1, 2, \cdots, 9, a, b, c, d, e, f$ 来表示的. 每个十六进制数可以用 4 个数位来表示. 因此 $(1)_{16} = (0001)_2, (8)_{16} = (1000)_2$, $(f) = (1111)_2 = (15)_{10}$. 0.3 节将描述表示机器数的 MATLAB 命令`format hex`.

习题 0.2

1. 求基为 10 的下列整数的二进制表示:

   (a) 64; (b) 17; (c) 79; (d) 227.

2. 把以下基为 10 的数转换成二进制数, 用上面加一横的记法表示无穷二进制数:

   (a) 10.5; (b) $\frac{1}{3}$; (c) $\frac{5}{7}$; (d) 12.8; (e) 55.4; (f) 0.1.

3. 求出 π 的二进制表示中的前 15 个数码.

4. 把下列二进制数转换成以 10 为基的数:

(a) 1010101; (b) 1011.101; (c) $10111.\overline{01}$; (d) $110.\overline{10}$;
(e) $10.\overline{110}$; (f) $110.1\overline{101}$; (g) $10.0101\overline{101}$; (h) $111.\overline{1}$.

0.3 实数的浮点表示

本节提供了一个关于浮点数的计算机计算的模型. 有各种模型, 但是为了简便, 我们将选择一个特例来详细地叙述. 我们选择的模型就是所谓 IEEE 754 浮点标准. IEEE(电气和电子工程师协会) 对于建立行业标准有着积极的兴趣, 他们的浮点运算格式已经成为整个计算机行业中单精度和双精度运算的共同标准.

当使用有限精度的计算机存储单元来表示无限精度的实数时, 舍入误差是不可避免的. 尽管我们希望在很长的运算中产生的小误差对答案只会产生很小的影响, 但是在许多情形下这事实上是一种一厢情愿的想法. **即使像 Gauss 消去法或微分方程的解法这类简单算法都可能把微小的误差放大很多**. 实际上, 本书的主题是帮助读者认识到由于数字计算机造成的对微小误差的放大, 计算存在不可靠的风险, 以及懂得怎样避免或者使这种风险减到最小.

0.3.1 浮点格式

IEEE 标准包含一组实数的二进制表示法. **浮点数**(floating point number) 由三部分组成, 即**符号**(sign, + 或 −)、**尾数**(mantissa, 它包含一串有效数字) 以及**阶**(即指数, exponent). 这三部分合一起就表示计算机中的浮点数.

浮点数有 3 种常用的精度等级, 即单精度、双精度和加长精度 (它也称为长双精度). 浮点数在这 3 种格式里规定的数位的数目分别是 32、64 和 80, 每一部分中的数位划分如下表所示:

精度	符号	阶	尾数
单	1	8	23
双	1	11	52
长双	1	15	64

这三类精度的用法实质上是相同的. IEEE 浮点数的正规化形式是

$$\pm 1.bbb\cdots b \times 2^p, \tag{0.6}$$

其中 N 个 b 中的每一个或者是 0 或者是 1, 而 p 是一个 M 位二进制数表示的指数. 如 (0.6) 式所示的正规化意味着最前面的 (最左面的) 数必须是 1.

当一个二进制数作为正规浮点数存储时它是 "左恰当的" (left-justified), 即最左边的 1 刚好位于小数点的左边. 通过改变阶来调整位移. 例如, 十进制数 9 的二进

制表示是 1001, 应该以

$$+1.001 \times 2^3$$

这种形式存储. 这是因为要把最左边的 1 移到正确的位置必须移动 3 个数位即乘以 2^3.

具体来说, 在大多数讨论中, 我们将限于双精度格式. 除了不同的阶和尾数长 M 及 N 以外, 我们用同样的方法处理单精度和长双精度. 在由许多 C 编译程序和 MATLAB 使用的双精度中, $M = 11, N = 52$.

这时, 双精度数 1 等于

$$+1.\boxed{0000000000000000000000000000000000000000000000000000} \times 2^0,$$

它有框起来的 52 位尾数. 下一个比 1 大的浮点数等于

$$+1.\boxed{0000000000000000000000000000000000000000000000000001} \times 2^0$$

或者是 $1 + 2^{-52}$.

定义 0.1 **机器**ε(machine epsilon) *表示 1 与大于 1 的最小浮点数之间的差, 记为 $\varepsilon_{\text{mach}}$. 对于 IEEE 双精度浮点标准来说,*

$$\varepsilon_{\text{mach}} = 2^{-52}.$$

十进制数 $9.4 = (1001.\overline{0110})_2$, 其左恰当形式是

$$+1.\boxed{0010110011001100110011001100110011001100110011001100}110\cdots \times 2^3,$$

这里我们把尾数的前 52 位框了起来. 于是产生了新的问题: 如何把十进制数 9.4 的无限二进制数表示成有限个数位?

我们必须用某种方法来截断一个数, 且由此必然产生小的误差. 有一种方法称为**断位法**(chopping), 就是简单地把小数点右边 52 位以外的数位舍去. 这种约定是很简单, 但这种方法会产生偏差, 因为它总会使其结果趋于零.

另一种方法是**舍入法**(rounding). 基为 10 时, 如果下一位数字是 5 或大于 5, 则习惯上把数字进一位, 否则就舍去. 在二进制, 这对应于当数位是 1 时便进一位. 准确地说, 在双精度格式中重要的数位是小数点右边第 53 个数位, 即落在框外的第 1 个数位. 由 IEEE 标准提供的默认舍入技巧是: 当第 53 个数位是 1 时, 则把第 52 个数位加 1 [上舍入 (舍入进位)]; 当第 53 个数位是 0 时, 第 52 个数位不变 [下舍入 (舍去)]; 但有一个例外, 即如果接在第 52 个数位之后的是数位 $10000\cdots$, 刚好卡在舍入和舍去中间, 我们选择舍入还是舍去的依据就是要使最后的第 52 位数等于 0(这里我们仅讨论尾数, 因为符号不起作用).

为什么会有这种奇怪的例外情形呢? 除了这种情形, 该规则意味着要舍入为最接近原数的正规化浮点数——因此它的名字叫做最近舍入规则. 由舍入而产生的误差将等可能地上舍入或下舍入. 在这种例外情况中, 可以舍入到两个等距的浮点数. 因此应该采用一种非系统性偏好上舍入或下舍入的方法. 这是试图避免由于简单的倾向性舍入而在长的计算中产生不想要的微小偏差. 在这种平局情形下, 选择让最后的第 52 位数等于 0, 虽多少有点随意性, 但是至少它不表示偏好是选择进位还是舍去. 习题 0.3 的第 6 题便揭示出在平局情形下为什么要随意地选择 0.

1. IEEE 最近舍入规则

对于双精度而言, 如果二进制小数点右边的第 53 位数是 0, 那么就舍去 (在第 52 位后面截断); 如果第 53 位数是 1, 那么就进位 (第 52 位数码加 1), 除非 1 右边的所有已知数位全是 0(若是这种情形, 当且仅当第 52 位数是 1 时, 第 52 位数加 1).

例如前面讨论过的数 9.4, 二进制小数点右边第 53 位数是 1, 紧随其后的是其他非 0 数, 最近舍入规则便要进位, 即第 52 位数加 1. 因此用浮点数表示 9.4 就是

$$+1.\boxed{0010110011001100110011001100110011001100110011001101} \times 2^3. \qquad (0.7)$$

定义 0.2 *使用最近舍入规则, 用 $\mathrm{fl}(x)$ 来表示与 x 相关的 IEEE 双精度浮点数.*

在计算机四则运算中, 用数位串 $\mathrm{fl}(x)$ 来代替实数 x. 根据这个定义, $\mathrm{fl}(9.4)$ 就是二进制表达式 (0.7) 中的那个数. 我们在舍入步通过丢掉该数右端无限的尾部 $0.1100 \times 2^{-52} \times 2^3 = 0.\overline{0110} \times 2^{-51} \times 2^3 = 0.4 \times 2^{-48}$, 再加上 $2^{-52} \times 2^3 = 2^{-49}$ 而得到浮点数表达式. 因此,

$$\mathrm{fl}(9.4) = 9.4 + 2^{-49} - 0.4 \times 2^{-48} = 9.4 + (1 - 0.8)2^{-49} = 9.4 + 0.2 \times 2^{-49}. \qquad (0.8)$$

换言之, 计算机使用双精度表达式和最近舍入规则在存储 9.4 时产生的误差为 0.2×2^{-49}. 我们称 0.2×2^{-49} 为**舍入误差**(rounding error).

重要的信息是用浮点数表示 9.4 并不等于 9.4, 尽管已很接近. 为了量化其接近程度, 我们使用误差的标准定义.

定义 0.3 *令 x_c 是准确值 x 的计算结果, 那么*

$$\textbf{绝对误差} = |x_c - x|, \qquad \textbf{相对误差} = \frac{|x_c - x|}{|x|},$$

假如后者存在.

2. 相对舍入误差

在 IEEE 计算机四则运算模型中, $\mathrm{fl}(x)$ 的相对舍入误差不大于机器 ε 的一半:

$$\frac{|\text{fl}(x)-x|}{|x|} \leqslant \frac{1}{2}\varepsilon_{\text{mach}}. \tag{0.9}$$

在数 $x=9.4$ 这种情形下，我们在 (0.8) 式中产生的舍入误差必须满足式 (0.9)：

$$\frac{|\text{fl}(9.4)-9.4|}{9.4} = \frac{0.2\times 2^{-49}}{9.4} = \frac{8}{47}\times 2^{-52} < \frac{1}{2}\varepsilon_{\text{mach}}.$$

例 0.2 对 $x=0.4$，求出双精度表达式 $\text{fl}(x)$ 及其舍入误差.

因为 $(0.4)_{10}=(0.\overline{0110})_2$，对齐左边得到

$$\begin{aligned}0.4 &= 1.10\overline{0110}\times 2^{-2}\\ &= +1.\boxed{1001100110011001100110011001100110011001100110011001}\\ &\quad 100110\cdots\times 2^{-2}.\end{aligned}$$

因此，按照舍入规则，$\text{fl}(0.4)$ 等于

$$+1.\boxed{1001100110011001100110011001100110011001100110011010}\times 2^{-2}.$$

这里第 52 位数加上 1，由于进行二进制加法使得第 51 位数也改变了.

仔细分析一下，我们在截断时丢掉了 $2^{-53}\times 2^{-2}+0.\overline{0110}\times 2^{-54}\times 2^{-2}$，而在舍入时又加上了 $2^{-52}\times 2^{-2}$. 因此

$$\begin{aligned}\text{fl}(0.4) &= 0.4-2^{-55}-0.4\times 2^{-56}+2^{-54}\\ &= 0.4+2^{-54}\left(-\frac{1}{2}-0.1+1\right)\\ &= 0.4+2^{-54}(0.4)\\ &= 0.4+0.1\times 2^{-52}.\end{aligned}$$

注意，对于 0.4，舍入时相对误差是 $\frac{0.1}{0.4}\times\varepsilon_{\text{mach}}=\frac{1}{4}\times\varepsilon_{\text{mach}}$，这符合式 (0.9). ◀

0.3.2 机器表示

到目前为止，我们已抽象地描述了浮点表示. 关于这种表示在计算机上是如何实现的，这里再作一些比较详细的讨论. 另外，本节将讨论双精度格式，至于其他格式，则十分类似.

对每个双精度浮点数指定 8 个字节或 64 位数存储其 3 个部分. 它们具有形式

$$\boxed{se_1e_2\cdots e_{11}b_1b_2\cdots b_{52}}\,, \tag{0.10}$$

其中 s 表示正负号，随后的 11 位数字表示阶，而小数点后面的 52 位数字表示尾数. 这里的符号码为 0 表示正数，符号码为 1 表示负数. 其中表示阶的 11 位数字是正

的二进制整数, 该整数经过由阶加上 $2^{10}-1=1\ 023$ 而得到, 而阶至少在 $-1\ 022$ 和 1 023 之间. 这样 $e_1,\cdots,e_{11}$ 的值在 1 到 2 046 之间, 而留下 0 与 2 047 另作别用, 后面我们将再考虑.

数 1023 称为双精度格式的**阶偏移**(exponent bias). 我们用它来把正、负的阶都转换成正的二进制数, 以便存储在表示阶的数位里. 对于单精度和长双精度来说, 阶偏移值分别是 127 和 16 383.

MATLAB 的`format hex`命令只是把 64 位机器数 (0.10) 表示成 16 位十六进制数, 即基是 16 的数. 因此, 前面的 3 位十六进制数表示符号和阶, 而后面的 13 位就是尾数.

例如, 一旦把 1 023 加到阶里, 数 1, 或者

$$1=+1.\boxed{0000000000000000000000000000000000000000000000000000}\times 2^0,$$

就有双精度机器数形式

0	01111111111	0000000000000000000000000000000000000000000000000000

前面的 3 个十六进制数字对应于

$$001111111111=3ff,$$

因此, 浮点数 1 的`format hex`表达式就是 $3ff0000000000000$. 可以在 MATLAB 中键入`format hex`命令并且输入数字 1 来检查这一结果.

例 0.3 求实数 9.4 的十六进制机器数表示式.

根据 (0.7) 式, 我们得到符号 $s=0$, 阶是 3, 小数点后的 52 位尾数是

0010	1100	1100	1100	1100	1100	1100	1100	1100	1100	1100	1100	1101

$\rightarrow (2ccccccccccccd)_{16}$

把 1 023 加到阶上, 得 $1\ 026=2^{10}+2$, 或者 $(10000000010)_2$. 把符号和阶结合起来, 就是 $(010000000010)_2=(402)_{16}$, 得到十六进制格式 $4022ccccccccccccd$. ◀

现在回到阶的特殊值 0 和 2 047. 当尾数数码串全是零或 NaN(Not a Number) 时 2047 表示 ∞. $e_1\cdots e_{11}=0$ 可以理解为非正规化的浮点形式

$$\pm 0.\boxed{b_1b_2\cdots b_{52}}\times 2^{-1\ 022}. \tag{0.11}$$

即最左边的数码不再假定是 1. 这些非正规化数叫做**次正规化**浮点数. 它们通过增加量级把很小的数的范围扩大了. 因此, 在双精度中, $2^{-52}\times 2^{-1\ 022}=2^{-1\ 074}$ 是最小的可表示数. 它的机器代码是

0	00000000000	0000000000000000000000000000000000000000000000000001

一定要知道最小的可表示数 $2^{-1\,074}$ 和 $\varepsilon_{\text{mach}} = 2^{-54}$ 之间的区别. 许多比 $\varepsilon_{\text{mach}}$ 小的数都是机器可表示的, 尽管把它们加到 1 上去, 可能没有什么影响. 另一方面, 完全不可能表示小于 $2^{-1\,074}$ 的数.

次正规数包括了最重要的数 0. 实际上, 次正规化表示式包括两个不同的浮点数: +0 和 −0. 它们被当作同一个实数看待. +0 的机器表示式带有符号码 $s = 0$, 阶码 $e_1 \cdots e_{11} = 00000000000$ 及尾数 52 个零, 简言之, 64 位数全是零. +0 的十六进制格式是 0000000000000000. 至于数 −0, 除了符号码 $s = 1$ 之外, 其余都相同, −0 的十六进制格式是 8000000000000000.

0.3.3 浮点数的加法

机器加法首先把两个要加的数字的小数点对齐, 然后将二者相加, 再把结果作为浮点数存储起来. 加法本身可以用更高的精度进行 (超过 52 位数), 这是因为加法是在专门用于此目的的寄存器内进行的. 为了存储为机器数, 相加之后的结果必须将二进制小数点之后部分舍入到 52 位数码.

例如, 把 1 加到 2^{-53} 便得到

$$
\begin{aligned}
& 1.\boxed{00\cdots0}\times 2^0 + 1.\boxed{00\cdots0}\times 2^{-53} \\
=& 1.\boxed{0000000000000000000000000000000000000000000000000000}\times 2^0 \\
& +0.\boxed{0000000000000000000000000000000000000000000000000000}1 \times 2^0 \\
=& 1.\boxed{0000000000000000000000000000000000000000000000000000}1 \times 2^0
\end{aligned}
$$

按照舍入规则, 这就存储为 $1. \times 2^0 = 1$. 因此在双精度 IEEE 计算中, $1 + 2^{-53}$ 就等于 1. 注意, 2^{-53} 是具有这种性质的最大浮点数; 在计算机四则运算中, 任何比它大的数加 1 的结果比 1 要大.

$\varepsilon_{\text{mach}} = 2^{-52}$ 并不表示在 IEEE 模型中可以忽略比 $\varepsilon_{\text{mach}}$ 小的数. 只要它们在模型中可以表示出来, 假定没有把它们与单位大小的数相加减, 那么用这样大的数进行的计算就如同精确的一样.

因为计算机在四则运算过程中会产生截断和舍入, 所以有时会给出出人意料的结果. 例如, 按 IEEE 最近舍入规则要求一台双精度计算机先存储 9.4, 然后减去 9, 再减去 0.4, 其结果将不是 0! 计算过程如下: 首先把 9.4 存储为 $9.4 + 0.2 \times 2^{-49}$, 这一点前面已说明过. 在减去 9 时 (注意 9 能被表示为不带误差), 其结果是 $0.4 + 0.2 \times 2^{-49}$. 现在再要求计算机减去 0.4, 结果减去了 (如我们在例 0.2 中看到的) 机器数 $\text{fl}(0.4) = 0.4 + 0.1 \times 2^{-52}$, 这样便留下

$$0.2 \times 2^{-49} - 0.1 \times 2^{-52} = 0.1 \times 2^{-52}(2^4 - 1) = 3 \times 2^{-53},$$

而不是零. 这是一个 $\varepsilon_{\mathrm{mach}}$ 量级的小的数, 但它不是零. 因为 MATLAB 的基本数据类型是 IEEE 双精度数, 我们可以在下面的 MATLAB 中说明这一结果:

```
>> format long
>> x=9.4

x =

   9.40000000000000

>> y=x-9

y =

   0.40000000000000

>> z=y-0.4

z =

     3.330669073875470e-16

>> 3*2^(-53)

ans =

     3.330669073875470e-16
```

例 0.4 求双精度浮点和 $(1+3\times 2^{-53})-1$.

当然, 按实数四则运算, 答案是 3×2^{-53}, 但是浮点四则运算的结果可能不同. 注意到 $3\times 2^{-53}=2^{-52}+2^{-53}$. 第一步加法是

$$
\begin{aligned}
&1.\boxed{00\cdots 0}\times 2^0+1.\boxed{10\cdots 0}\times 2^{-52}\\
=&1.\boxed{0000000000000000000000000000000000000000000000000000}\times 2^0\\
&+0.\boxed{0000000000000000C0000000000000000000000000000000001}1\times 2^0\\
=&1.\boxed{0000000000000000000000000000000000000000000000000001}1\times 2^0
\end{aligned}
$$

这对于舍入规则而言又是一种例外情形. 因为和中的第 52 位数是 1, 我们必须上舍入, 也就是要在第 52 位数上加 1. 这样便得到

$$
+1.\boxed{0000000000000000000000000000000000000000000000000010}\times 2^0,
$$

它等于 $1+2^{-51}$. 因此, 减去 1 以后的结果是 2^{-51}. 它等于 $2\varepsilon_{\mathrm{mach}}=4\times 2^{-53}$. 再次注意计算机四则运算与准确四则运算之间的差别. 用MATLAB检验这一结果. ◀

MATLAB 中的计算或者按照 IEEE 标准进行浮点计算的任何编译程序中的计算都遵照了本节所描述的精度规则, 尽管浮点计算的结果可能因为它不同于准确的四则运算而给出意外的结果. 但它总是可以预测的. 最近舍入规则是一种典型的默认舍入, 但我们可以根据需要通过使用编译程序标志改变成其他的舍入规则. 对不同的舍入规则得到的结果进行比较, 有时可作为评估计算稳定性的非正规方法.

你很可能会感到意外, 因为就凭相对于 $\varepsilon_{\text{mach}}$ 都小的舍入误差, 就可使有意义的计算溢出. 为此 0.4 节将介绍一种机制. 更一般地, 误差的放大和适用条件将是第 1 章、第 2 章及其他地方反复研究的主题.

习题 0.3

1. 把下列基为 10 的数转换成二进制, 并用最近舍入规则把每个数表示成浮点数 $\text{fl}(x)$:
   (a) $\frac{1}{4}$; **(b)** $\frac{1}{3}$; **(c)** $\frac{2}{3}$; **(d)** 0.9.
2. 把下列基为 10 的数转换成二进制, 并用最近舍入规则把每个数表示成浮点数 $\text{fl}(x)$:
   (a) 9.5; **(b)** 9.6; **(c)** 100.2; **(d)** $\frac{44}{7}$.
3. 使用最近舍入规则, 依照 IEEE 双精度计算机四则运算规则, 手工求下列各式 (用 MATLAB 检验你的结果):
   (a) $\left(1+(2^{-51}+2^{-53})\right)-1$;
   (b) $\left(1+(2^{-51}+2^{-52}+2^{-53})\right)-1$.
4. 使用最近舍入规则, 依照 IEEE 双精度计算机四则运算规则, 手工求下列各式:
   (a) $\left(1+(2^{-51}+2^{-52}+2^{-54})\right)-1$;
   (b) $\left(1+(2^{-51}+2^{-52}+2^{-60})\right)-1$.
5. 用 MATLAB 的`format hex`格式写出下列每一个数. 给出结果, 并用 MATLAB 检验你的答案:
   (a) 8; **(b)** 21; **(c)** $\frac{1}{8}$; **(d)** $\text{fl}(\frac{1}{3})$; **(e)** $\text{fl}(\frac{2}{3})$;
   (f) $\text{fl}(0.1)$; **(g)** $\text{fl}(-0.1)$; **(h)** $\text{fl}(-0.2)$.
6. 在双精度浮点四则运算中, 使用 IEEE 最近舍入规则, $\frac{1}{3}+\frac{2}{3}$ 是否恰好等于 1? 你需要用习题 1 中的 $\text{fl}(\frac{1}{3})$ 和 $\text{fl}(\frac{2}{3})$. 这是否有助于解释规则正如它所表述的那样? 如果不用 IEEE 舍入规则而是把第 52 位数之后截去, 这个和还相同吗?
7. **(a)** 通过计算 $\left(\frac{7}{3}-\frac{4}{3}\right)-1$, 解释为什么在计算机上使用 IEEE 双精度和 IEEE 最近舍入规则就能确定机器 ε.
   (b) $\left(\frac{4}{3}-\frac{1}{3}\right)-1$ 也给出 $\varepsilon_{\text{mach}}$ 吗? 通过转换到浮点数. 并进行这种计算机四则运算来解释.
8. 使用最近舍入规则, 在双精度浮点四则运算中确定 $1+x>1$ 是否成立:
   (a) $x=2^{-53}$; **(b)** $x=2^{-53}+2^{-60}$.
9. 对 IEEE 计算机加法来说, 结合律是否成立?
10. 求出 IEEE 双精度表示式 $\text{fl}(x)$, 并对所给实数求出准确的差 $\text{fl}(x)-x$. 校验其相对舍入误差不大于 $\frac{1}{2}\varepsilon_{\text{mach}}$.
    (a) $x=2.75$; **(b)** $x=2.7$; **(c)** $x=\frac{10}{3}$.

11. 用计算器或者用 MATLAB 计算 $\sin(10^{30})$, 你能保证多少位正确的数字? 对 $\sin(10^{30}+1)$ 呢? 解释你的理由. 在双精度计算机上还有没有方法使计算这个数更精确?

0.4 有效数字的损失

详细了解计算机的四则运算有助于我们理解计算机计算中的潜在失误. 以多种形式出现的一个主要问题是, 当两个几乎相等的数相减时会损失有效数字. 就其最简单形式而言, 问题是很明显的. 例如, 减法问题

$$\begin{array}{r} 123.4567 \\ -123.4566 \\ \hline 0.0001 \end{array}$$

刚开始时两个数都有 7 位精确数字, 而结束时结果仅有 1 位精确数字. 尽管这个例子十分简单明了, 但还有其他更难捉摸的有效数字损失的例子, 并且在许多情形下, 这可以通过重构计算而避免.

例 0.5 在 3 位十进制数字计算机上计算 $\sqrt{9.01}-3$.

这个例子还是相当简单, 提出来仅仅是为了说明上述思想. 我们假定使用 3 位十进制数字计算机, 而不是如按双精度 IEEE 标准格式使用带有 52 位尾数的计算机. 使用 3 位数字计算机意味着, 存储的每一步中间计算结果都是带有 3 位尾数的浮点数. 例题中给定的数据 (9.01 与 3.00) 具有 3 位数字精度. 因为我们要用 3 位数字计算机, 所以最多人们可能希望得到的答案是准确到 3 位数字 (因为在计算中仅仅有 3 位数字, 当然我们不能期望得到更多). 在计算器上检验, 我们看到准确答案近似地等于 $0.001\,662 = 1.666\,2\times10^{-3}$. 我们用 3 位数字计算机得到了多少位正确的数字?

事实表明, 没有 1 个正确的数字. 因为 $\sqrt{9.01}\approx 3.001\,666\,2$. 当把这个中间结果存储成 3 位有效数字, 就得到 3.00. 减去 3.00 以后, 得到最后的答案是 0.00, 没有一个有效数字是正确的.

想不到的是, 即使在 3 位数字计算机上, 也有一种方法可以挽救这一计算. 造成有效数字损失的原因是我们明显地把两个几乎相等的数 $\sqrt{9.01}$ 和 3 相减. 用代数重新改写表示式

$$\begin{aligned}\sqrt{9.01}-3 &= \frac{(\sqrt{9.01}-3)(\sqrt{9.01}+3)}{\sqrt{9.01}+3} = \frac{9.01-3^2}{\sqrt{9.01}+3} \\ &= \frac{0.01}{3.00+3} = \frac{0.01}{6} = 0.001\,67 \approx 1.67\times10^{-3}\end{aligned}$$

就能避免这个问题.

这里, 我们把尾数的最后一个数字四舍五入为 7, 这是因为它后面一个数字是 6. 注意, 使用这种方法得到的 3 位数字都正确, 至少正确答案舍入到的 3 位数字是正确的. 教训是, 如果有可能, 应寻找一种避免很接近的数相减的方法, 这是很重要的. ◀

上述例题所示范的方法实质上是一种技巧. 乘以 "共轭式" 往往能帮助重构计算. 通常可以使用如三角公式那样的特殊恒等式. 例如, 当 x 接近零时, 计算 $1-\cos x$ 就会导致有效数字的丢失. 我们针对某一范围的输入数 x 来比较下面两个表达式的计算:

$$E_1=\frac{1-\cos x}{\sin^2 x} \quad 和 \quad E_2=\frac{1}{1+\cos x}.$$

我们通过对 E_1 的分子和分母都乘以 $1+\cos x$, 并利用三角恒等式 $\sin^2 x+\cos^2 x=1$ 就得到了 E_2. 这两个表达式就无限精度而言都是相等的. 采用双精度 MATLAB 计算, 我们得到表 0-1. 右边 E_2 列给出了可以正确到的最多数位. 计算 E_1 时, 由于相近的数相减, 因此当 $x<10^{-5}$ 时, 便会出现严重的问题; 而当 $x\leqslant 10^{-8}$ 时就不再有正确的有效数字.

表 0-1

x	E_1	E_2
1.000 000 000 000 00	0.649 223 205 204 76	0.649 223 205 204 76
0.100 000 000 000 00	0.501 252 086 288 58	0.501 252 086 288 57
0.010 000 000 000 00	0.500 012 500 208 48	0.500 012 500 208 34
0.001 000 000 000 00	0.500 000 124 992 19	0.500 000 125 000 02
0.000 100 000 000 00	0.499 999 998 627 93	0.500 000 001 250 00
0.000 010 000 000 00	0.500 000 041 386 85	0.500 000 000 012 50
0.000 001 000 000 00	0.500 044 450 291 34	0.500 000 000 000 13
0.000 000 100 000 00	0.499 600 361 081 32	0.500 000 000 000 00
0.000 000 010 000 00	0.000 000 000 000 00	0.500 000 000 000 00
0.000 000 001 000 00	0.000 000 000 000 00	0.500 000 000 000 00
0.000 000 000 100 00	0.000 000 000 000 00	0.500 000 000 000 00
0.000 000 000 010 00	0.000 000 000 000 00	0.500 000 000 000 00
0.000 000 000 001 00	0.000 000 000 000 00	0.500 000 000 000 00

对于 $x=10^{-4}$, 表达式 E_1 已经有几位不正确的数字, 而且当 x 更小时情况更糟. 等价的表达式 E_2 没有相近数相减. 因此不存在这类问题.

二次公式经常会牵涉到有效数字的损失. 同样, 只要你知道如何重构表达式就容易避免.

例 0.6 求二次方程 $x^2+9^{12}x=3$ 的两个根.

譬如用 MATLAB 按双精度四则运算来求解. 除非你得知有效数字会失去并且知道如何克服它, 否则没有人能给出正确答案. 问题是求两个根, 假设要有 4 位数

字是精确的. 到目前为止这似乎是一个容易的问题. 形如 $ax^2+bx+c=0$ 的二次方程的根由以下二次公式给出:

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}. \tag{0.12}$$

对我们的问题而言, 即是

$$x=\frac{-9^{12}\pm\sqrt{9^{24}+4(3)}}{2}.$$

取负号给出根

$$x_1=-2.824\times10^{11},$$

它准确到 4 位有效数字. 取正号的根

$$x_2=\frac{-9^{12}+\sqrt{9^{24}+4(3)}}{2},$$

MATLAB 计算得到 0. 尽管准确答案是接近零, 但是这个答案没有正确的有效数字——即使定义该问题的数是精确给定的 (本质上有无穷多位准确数字), 而且事实上 MATLAB 可进行大约 16 位有效数字的计算 (依据是, MATLAB 的机器 $\varepsilon_{\text{mach}}$ 是 $2^{-52}\approx2.2\times10^{-16}$). 我们该如何解释求 x_2 的准确数字完全失败了?

答案是有效数字的损失. 显然, 相对而言, 9^{12} 和 $\sqrt{9^{24}+4(3)}$ 几乎相等. 更精确地说, 作为存储的浮点数, 它们的尾数不但前几位相似, 而且近乎恒等. 当它们按照二次求根公式相减时, 当然结果是零.

能否挽救这个计算? 我们必须正视有效数字损失的问题. 计算 x_2 的正确方法是重构二次求根公式:

$$\begin{aligned}x_2&=\frac{-b+\sqrt{b^2-4ac}}{2a}=\frac{(-b+\sqrt{b^2-4ac})(b+\sqrt{b^2-4ac})}{2a(b+\sqrt{b^2-4ac})}\\&=\frac{-4ac}{2a(b+\sqrt{b^2-4ac})}=-\frac{2c}{b+\sqrt{b^2-4ac}}.\end{aligned}$$

对于我们的例子, 只要代入 a,b,c, 按照 MATLAB 就可得到 $x_2=1.062\times10^{-11}$, 它具有要求精度的 4 位有效数字. ◀

这个例子告诉我们: 在 a 或者 (以及)c 相对于 b 较小的情形中, 使用二次求根公式 (0.12) 必须小心. 更精确地说, 当 $4|ac|\ll b^2$ 时, b 和 $\sqrt{b^2-4ac}$ 几乎大小相等, 其中一个根牵涉到有效数字的损失. 在这种情形下, 如果 b 是正数, 那么两个根应如下计算:

$$x_1=-\frac{b+\sqrt{b^2-4ac}}{2a},\quad x_2=-\frac{2c}{b+\sqrt{b^2-4ac}}. \tag{0.13}$$

注意, 上述两个公式都不会受到几乎相等的数相减的影响. 另一方面, 如果 b 是负数, 并且 $4|ac| \ll b^2$, 那么两个根最好如下计算:

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}, \quad x_2 = \frac{2c}{-b+\sqrt{b^2-4ac}}. \tag{0.14}$$

习题 0.4

1. 确定对 x 的哪个值会产生几乎相等的数相减, 并求一种变形以避免这个问题:
   (a) $\frac{1-\sec x}{\tan^2 x}$; (b) $\frac{1-(1-x)^3}{x}$; (c) $\frac{1}{1+x}-\frac{1}{1-x}$.
2. 求方程 $x^2+3x-8^{-14}=0$ 的根, 要求 3 位数字的精度.
3. 说明如何最精确地计算方程 $x^2+bx-10^{-12}=0$ 的两个根, 其中 b 是大于 100 的数.
4. 证明公式 (0.14).

计算机问题 0.4

1. 对于 $x=10^{-1},\cdots,10^{-14}$, 按照双精度四则运算, 计算下面的表达式 (譬如用 MATLAB). 然后用不会让两个几乎相等的数相减的另一种形式, 再计算一遍, 并列结果表. 说出对每一个 x, 原来表达式中准确数字的位数:
   (a) $\frac{1-\sec x}{\tan^2 x}$; (b) $\frac{1-(1-x)^3}{x}$.
2. 求 p 的最小值, 使得用双精度四则运算计算下列表达式在 $x=10^{-p}$ 处没有准确的有效数字 (提示: 先求当 $x\to 0$ 时表达式的极限):
   (a) $\frac{\tan x - x}{x^3}$; (b) $\frac{e^x+\cos x-\sin x-2}{x^3}$.
3. 考虑两直角边的边长分别是 3 344 556 600 和 1.222 222 2 的直角三角形. 问斜边相比较长的直角边长多少? 给出至少有 4 位准确数字的答案.

0.5 微积分回顾

以后将需要微积分的某些重要基础知识. 介值定理和中值定理对第 1 章中的方程求解很重要. Taylor 定理对理解第 3 章中的插值很重要, 并对第 6~8 章中微分方程的求解也是至关重要的.

连续函数的图形没有间断. 例如, 若某函数对某一 x 值为正而对另一 x 值为负, 那么它必在某处为零. 这个事实是第 1 章方程解法的基础. 下面第一个定理阐述了这一思想.

定理 0.4(介值定理) *设 f 是区间 $[a,b]$ 上的一个连续函数, 那么 f 取到 $f(a)$ 和 $f(b)$ 之间的任一个值. 更精确地说, 如果 y 是 $f(a)$ 和 $f(b)$ 之间的一个数, 那么存在一个数 $c(a\leqslant c\leqslant b)$ 使得 $f(c)=y$.*

例 0.7 证明 $f(x)=x^2-3$ 在区间 $[1,3]$ 上必取到值 0 和 1.

因为 $f(1)=-2, f(3)=6$, 所有在 -2 和 6 之间的值, 包含 0 和 1, 均可被 f 取到. 例如, 取 $c=\sqrt{3}$, 则有 $f(c)=f(\sqrt{3})=0$, 其次又有 $f(2)=1$. ◀

定理 0.5(连续函数的极限) *设 f 是在 x_0 的某一邻域内的连续函数, 并且 $\lim_{n\to\infty} x_n = x_0$, 那么*

$$\lim_{n\to\infty} f(x_n) = f\left(\lim_{n\to\infty} x_n\right) = f(x_0).$$

换言之, 极限可以取到连续函数里边.

定理 0.6(中值定理) *设 f 是在区间 $[a,b]$ 上的连续可微函数, 那么在 a 和 b 之间存在一个数 c, 使得 $f'(c) = \frac{f(b)-f(a)}{b-a}$.*

例 0.8 在区间 [1, 3] 上, 对 $f(x) = x^2 - 3$ 应用中值定理.

因为 $f(1) = -2$, $f(3) = 6$, 中值定理的内容是在区间 $[1,3]$ 中存在数 c 满足 $f'(c) = \frac{6-(-2)}{3-1} = 4$. 容易求出这样的 c. 由于 $f'(x) = 2x$, 因此正确的 $c = 2$. ◀

下面的陈述是中值定理的一个简单推论.

定理 0.7(Rolle 定理) *设 f 是在区间 $[a,b]$ 上的连续可微函数, 并假定 $f(a) = f(b)$, 那么在 a 和 b 之间存在一个数 c, 使得 $f'(c) = 0$.*

如果已知 f 在点 x 处的值, 那么就能获得附近点的许多信息. 例如, 假如 $f'(x) > 0$, 那么, 在右边的邻近点处 f 有较大的值, 而在左边的邻近点处有较小的值. Taylor 定理利用在 x 处的导数给出了在 x 的小邻域内函数值的完全估计.

定理 0.8(带余项的 Taylor 定理) *设 x 和 x_0 是实数, f 在区间 $[x_0, x]$(或 $[x, x_0]$)上$k+1$ 次连续可微, 那么在 x 与 x_0 之间存在一个数 c, 使得*

$$f(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \frac{(x-x_0)^3}{3!}f'''(x_0) \\ + \cdots + \frac{(x-x_0)^k}{k!}f^{(k)}(x_0) + \frac{(x-x_0)^{k+1}}{(k+1)!}f^{(k+1)}(c).$$

上式中的多项式部分, 即 $(x - x_0)$ 的 k 次项之前部分, 称为 f 在以 x_0 为中心的邻域上的**k次Taylor多项式**. 最后一项称为**Taylor余项**. 在 Taylor 余项是一个小量的情形下, Taylor 定理给出了一种方法, 用多项式去近似一般的光滑函数. 在用计算机求解问题时, 这是非常方便的. 诚如前面提到的, 能非常有效地计算多项式.

例 0.9 求 $f(x) = \sin x$ 在以点 $x_0 = 0$ 为中心的邻域上的 4 次 Taylor 多项式 $P_4(x)$. 使用 $P_4(x)$ 来计算 $\sin x$ 在 $|x| \leqslant 0.000\,1$ 时的值, 估计最大可能误差.

容易求出多项式是 $P_4(x) = x - \frac{x^3}{6}$. 注意 4 次项没有出现, 这是因为它的系数为 0. 余项是

$$\frac{x^5}{120}\cos c,$$

它的绝对值不可能大于 $\frac{|x|^5}{120}$. 对于 $|x| \leqslant 0.000\,1$, 这个余项最多是 $\frac{10^{-20}}{120}$, 譬如在双精度下用 $x - \frac{x^3}{6}$ 来近似代替 $\sin 0.000\,1$ 将看不到上述余项. 可在 MATLAB 的计算中检验这一结论. ◀

最后给出中值定理的积分形式:

定理 0.9(积分形式的中值定理) 设 f 是区间 $[a,b]$ 上的连续函数, g 是可积函数, 并且在 $[a,b]$ 上不变号, 那么在 $[a,b]$ 内存在一个数 c, 使得

$$\int_a^b f(x)g(x)\mathrm{d}x = f(c)\int_a^b g(x)\mathrm{d}x.$$

习题 0.5

1. 用介值定理证明: 存在 c 且 $0<c<1$ 使 $f(c)=0$.

(a) $f(x)=x^3-4x+1$; **(b)** $f(x)=5\cos\pi x-4$; **(c)** $f(x)=8x^4-8x^2+1$.

2. 对于在区间 [0, 1] 上的 $f(x)$, 求满足中值定理的 c.

(a) $f(x)=\mathrm{e}^x$; **(b)** $f(x)=x^2$; **(c)** $f(x)=\frac{1}{x+1}$.

3. 对于区间 [0,1] 上的 $f(x),g(x)$, 求满足积分形式中值定理的 c.

(a) $f(x)=x, g(x)=x^2$; **(b)** $f(x)=x^2, g(x)=x$; **(c)** $f(x)=x, g(x)=\mathrm{e}^x$.

4. 求以下函数在点 $x=0$ 处的二次 Taylor 多项式:

(a) $f(x)=\mathrm{e}^{x^2}$; **(b)** $f(x)=\cos 5x$; **(c)** $f(x)=\frac{1}{x+1}$.

5. 求以下函数在点 $x=0$ 处的 5 次 Taylor 多项式:

(a) $f(x)=\mathrm{e}^{x^2}$; **(b)** $f(x)=\cos 2x$; **(c)** $f(x)=\ln(1+x)$; **(d)** $f(x)=\sin^2 x$.

6. **(a)** 求 $f(x)=x^{-2}$ 在点 $x=1$ 处的 4 次 Taylor 多项式;

(b) 用 (a) 的结果计算 $f(0.9)$ 和 $f(1.1)$ 的近似值;

(c) 利用 Taylor 余项, 求 Taylor 多项式的误差公式. 对 (b) 中得到的两个近似值给出误差界. 你认为 (b) 中两个近似值哪一个更接近准确值?

(d) 在每一种情形下, 用计算器把实际误差与 (c) 中得到的误差界进行比较.

7. 对 $f(x)=\ln x$, 做习题 6(a)–6(d).

8. **(a)** 求 $f(x)=\cos x$ 在以 $x=0$ 为中心的邻域上的 5 次 Taylor 多项式 $P(x)$;

(b) 当 x 在 $\left[-\frac{\pi}{4},\frac{\pi}{4}\right]$ 中时, 用 $P(x)$ 近似代替 $f(x)=\cos x$, 求其误差的上界.

9. 当 x 是小量时, $\sqrt{1+x}$ 通常的近似是 $1+\frac{1}{2}x$. 用带有余项的 $f(x)=\sqrt{1+x}$ 的一次 Taylor 多项式去求形如 $\sqrt{1+x}=1+\frac{1}{2}x\pm E$ 的公式. 并就近似 $\sqrt{1.02}$ 的情形, 计算 E 的值. 用计算器比较实际误差与你的误差界 E.

软件和进一步阅读

浮点运算的 IEEE 标准可以在 [3] 中找到. 原始资料 [1, 6] 相当详细地讨论了浮点运算, 新近出版的 [5] 强调了 IEEE 754 标准. Wilkinson 的著作 [7] 与 Knuth 的著作 [4] 对硬件和软件的发展有重大的影响.

有一些软件包主要用于一般目的的科学计算问题, 其中大部分都是浮点运算. 由 AT& T 贝尔实验室、田纳西大学、橡树岭国家实验室共同维护的 Netlib(http: //www.netlib.org) 收集了免费的软件. 这些软件包括用 Fortran, C 和 Java 编写的高质量的程序, 但几乎不提供技术支持. 代码中的注释对用户使用这些程序给出了足够充分的指导.

数值算法集团 (Numerical Algorithms Group, NAG)(http://www.nag.co.uk) 出售可解一般应用型数学问题的 1 400 多个用户可调用子程序的软件库. 这些程序中有以 Fortran 和 C 语言编写的, 也可从 Java 程序环境调用中. NAG 包括的程序库可用于共享存储器及分布式存储计算.

IMSL 数值软件库由美商威能信息公司 (Visual Numerics, INC.)(http://www.vni.com) 出品, 除与 NAG 软件库有类似的覆盖领域外, 它还提供可进行数据分析及可视化操作的强大的编程语言 PV−WAVE.

计算环境 Mathematica、Maple 及 MATLAB 都在发展, 包含了许多如前所述的相同的计算方法, 并提供了内置的编辑及图形接口. Mathematica(http://www.wolframresearch.com) 和 Maple(www.maplesoft.com) 由于具备新颖的符号计算功能而著名, MATLAB 通过将基本的高质量软件应用于各个不同的方面的 "工具箱" 而服务于科学与工程应用领域.

本书中, 我们常常利用 MATLAB 程序来说明基本算法, 所给的 MATLAB 代码只是用来说明算法而已. 为清晰简明及阅读方便起见, 速度与可靠性经常不予考虑. 对 MATLAB 不熟悉的读者应从附录 B 的指南出发, 相信很快就能学会编制自己的实现程序.

第1章 解 方 程

最近发掘出的楔形小片表明, 古巴比伦人在计算 2 的平方根时能够精确到 5 位小数. 他们的技巧尚无人知晓, 但本章将介绍他们可能使用过的迭代法, 而且这种迭代法仍被现代的计算器用于求平方根.

Stewart 平台, 原是邓禄普轮胎公司的 Eric Gough 在 20 世纪 50 年代开发的能精确定位的有 6 个自由度的机器人, 当时用于检查飞机轮胎. 它在制造大型飞行模拟器以及一些要求高精度的领域 (比如医学和外科手术) 中具有独特的优点. 解前向运动学问题需要确定在给定支柱长度时平台的位置和定向.

实例检验 本章末的实例检验 1 使用第 1 章中的方程解法, 用于解平面形式的 Stewart 平台的前向运动学问题.

方程求解是科学计算中最基本的问题之一. 本章介绍寻找方程 $f(x) = 0$ 的解 x 的许多迭代方法, 这些方法也具有十分重要的实用价值. 此外, 这些方法阐明了收敛性和复杂性在科学计算中的核心作用.

为什么需要知道不止一种方法来解方程? 方法的选择经常取决于计算函数 f 的值或者其导数值的代价. 假设 $f(x) = \mathrm{e}^x - \sin x$, 那么大约需要用千分之几秒就可以确定 $f(x)$, 并且如果需要的话, 它的导数也可求得. 如果 $f(x)$ 表示乙烯乙二醇溶液在 x 个大气压强下的冰点, 在具有良好设备的实验室中算每一个函数可能需用相当长的时间来计, 并且确定导数值可能办不到.

除了介绍诸如对分法、不动点迭代及 Newton 法外, 我们还将分析它们的收敛速度并讨论它们的计算复杂性. 稍后, 我们还将提出更复杂的方程解法, 包括结合了几种解法之长处的 Brent 方法.

1.1 对 分 法

如何在一本陌生的电话簿中查找名字? 要查找 “Smith”, 你可能一开始把书翻到你的最佳估计, 譬如说字母 Q. 下一步你可能一下子翻到字母 U 处. 现在你已经把名字 Smith“括” 了起来, 并且通过缩小括号范围来找到它, 这些括号最终会收敛于你要找的名字. 对分法代表了这类尽可能有效地执行的推理类型.

1.1.1 根隔离法

定义 1.1 如果 $f(r) = 0$, 那么函数 $f(x)$ 在 $x = r$ 处有一个根.

解方程的第一步是证明根存在. 确保这一点的一种方法是把根括住: 在实轴上找一个区间 $[a,b]$, 使数对 $\{f(a),f(b)\}$ 的一个数是正的, 而另一个是负的. 这可以表示为 $f(a)f(b)<0$. 如果 f 是连续函数, 那么将有一个根: 在 a 和 b 之间的一个数 r 使得 $f(r)=0$. 介值定理 0.4 的如下推论概括了这一事实.

定理 1.2 设 f 是区间 $[a,b]$ 上的连续函数, 满足 $f(a)f(b)<0$. 那么 f 在 a 和 b 之间有一个根, 即存在一个数 r 满足 $a<r<b$ 以及 $f(r)=0$.

在图 1-1 中, $f(0)f(1)=(-1)(1)<0$. 有一个根恰好在 0.7 的左边. 如何能把我们对根的位置的最初估计精确到更多位的小数?

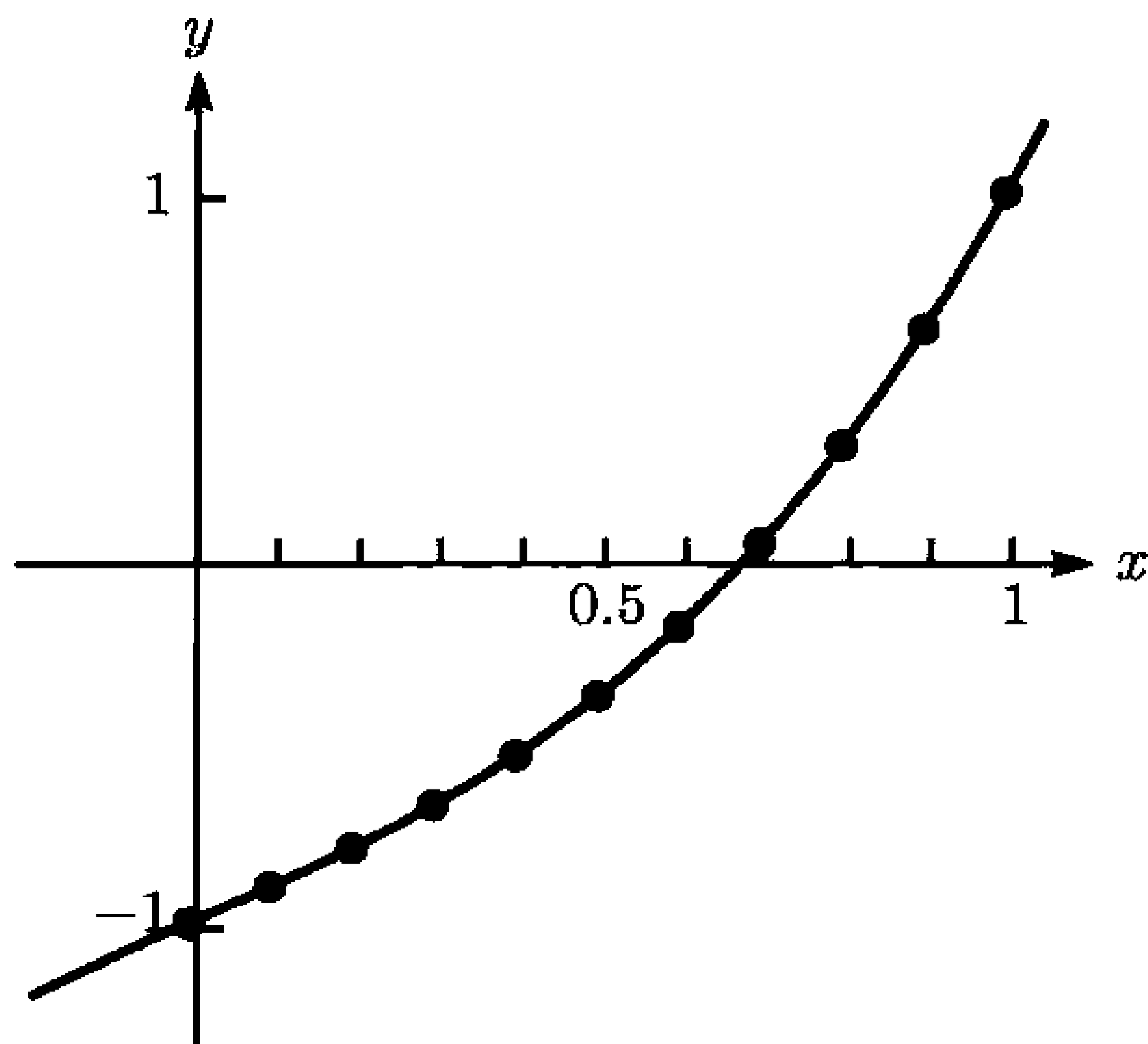


图 1-1 $f(x)=x^3+x-1$ 的图形: 函数在 0.6 和 0.7 之间有一个根

当给定函数图形时, 我们将从目测求解的方法中得到启发. 我们不可能从区间的左端点开始向右移动, 然后在根那里停止. 也许更好的方法是用目测先确定一个大概的位置, 譬如这个根到底是偏向于区间的左边还是右边, 然后更精确地确定根位于右边多远或者左边多远, 并且逐步提高精度, 正如在电话簿中查找名字一样. 如图 1-2 所示, 这种一般方法将在对分法中详细叙述.

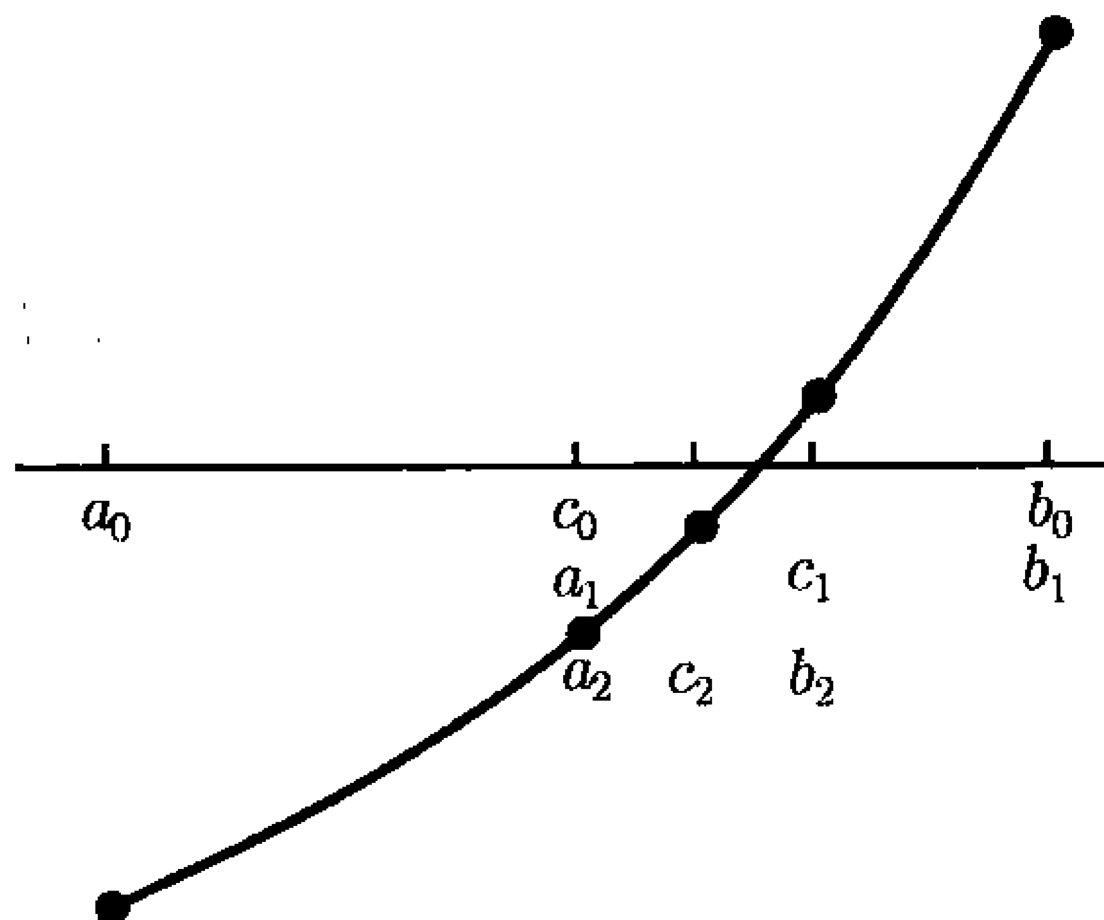


图 1-2 对分法: 第一步, 检查 $f(c_0)$ 的符号, 因为 $f(c_0)f(b_0)<0$, 所以取 $a_1=c_0,b_1=b_0$, 并且用右半区间 $[a_1,b_1]$ 代替原区间 $[a_0,b_0]$; 第二步, 用左半区间 $[a_2,b_2]$ 代替 $[a_1,b_1]$

对分法

给定初始区间 $[a,b]$, 使 $f(a)f(b)<0$

while $(b-a)/2 < \mathrm{TOL}$

　　$c=(a+b)/2$

　　if $f(c)=0$,**stop, end**

　　if $f(a)f(c)<0$

　　　　$b=c$

　　else

　　　　$a=c$

　　end

end

最后得到的区间 $[a,b]$ 包含一个根.

近似根是 $(a+b)/2$.

检查函数在区间的中点 $c=(a+b)/2$ 的值. 由于 $f(a)$ 和 $f(b)$ 反号, 所以不是 $f(c)=0$(这时已完成求根), 就是 $f(c)$ 的符号与 $f(a)$ 或 $f(b)$ 的符号相反. 如果 $f(c)f(a)<0$, 那么就能确定有一个根在区间 $[a,c]$ 中, 它的长度是原区间 $[a,b]$ 的一半. 如果 $f(c)f(b)<0$, 那么区间 $[c,b]$ 就有同样的性质. 无论哪种情形, 下一步就把问题化为在原区间长度一半的区间中求根. 重复这个步骤就能越来越精确地求得函数的根.

通过每一步用新的区间括住根, 当区间变小时根的位置的不确定度降低了. 我们不需要知道函数的全图. 我们已把求函数值的工作简化到那些必要的步骤.

例 1.1　在区间 $[0,1]$ 上, 用对分法求函数 $f(x)=x^3+x-1$ 的根.

正如先前指出的, $f(a_0)f(b_0)=(-1)(1)<0$, 所以在区间内存在一个根. 区间中点是 $c_0=\frac{1}{2}$. 第一步算出 $f\left(\frac{1}{2}\right)=-\frac{3}{8}<0$, 故选择新的区间 $[a_1,b_1]=\left[\frac{1}{2},1\right]$, 这是因为 $f(\frac{1}{2})f(1)<0$. 第二步算出 $f(c_1)=f(\frac{3}{4})=\frac{11}{64}>0$, 从而得到新的区间 $[a_2,b_2]=\left[\frac{1}{2},\frac{3}{4}\right]$. 继续使用这种方法得到表 1-1 所示的区间. 由该表格可以推断, 解

表　1-1

i	a_i	$f(a_i)$	c_i	$f(c_i)$	b_i	$f(b_i)$
0	0.000 0	−	0.500 0	−	1.000 0	+
1	0.500 0	−	0.750 0	+	1.000 0	+
2	0.500 0	−	0.625 0	−	0.750 0	+
3	0.625 0	−	0.687 5	+	0.750 0	+
4	0.625 0	−	0.656 2	−	0.687 5	+
5	0.656 2	−	0.671 9	−	0.687 5	+
6	0.671 9	−	0.679 7	−	0.687 5	+
7	0.679 7	−	0.683 6	+	0.687 5	+
8	0.679 7	−	0.681 6	−	0.683 6	+
9	0.681 6	−	0.682 6	+	0.683 6	+

被括在 $a_9 \approx 0.681\,6$ 和 $c_9 \approx 0.682\,6$ 之间. 这个区间的中点 $c_{10} \approx 0.682\,1$ 就是对根的最佳估计.

虽然问题是求根, 实际上我们求得的是包含根的区间 $[0.681\,6, 0.682\,6]$, 换言之, 根 $r = 0.682\,1 \pm 0.000\,5$. 它必须满足某种精度. 当然, 如果需要, 可以通过执行更多步的对分法来提高精度. ◀

在对分法的每一步中我们求出当前区间 $[a_i, b_i]$ 的中点 $c_i = (a_i + b_i)/2$, 计算 $f(c_i)$ 并且比较符号. 如果 $f(c_i)f(a_i) < 0$, 我们取 $a_{i+1} = a_i$ 及 $b_{i+1} = c_i$; 如果相反即 $f(c_i)f(a_i) > 0$, 那么取 $a_{i+1} = c_i$ 及 $b_{i+1} = b_i$. 每一步需要计算一次函数 $f(x)$ 的值, 并要对分包含根的区间, 把区间长度减半. 在计算 n 步的 c 和 $f(c)$ 之后, 我们已经进行 $n+1$ 次函数求值, 而且我们对解的最好估计是最后一个区间的中点. 这个算法可以写成以下的 MATLAB 代码.

```
%Program 1.1 Bisection Method
%Computes approximate solution of f(x)=0
%Input: inline function f, a,b such that f(a)*f(b)<0,
%        and tolerance tol
%Output: Approximate solution xc
function xc=bisect(f,a,b,tol)
if sign(f(a))*sign(f(b)) >= 0
  error('f(a)f(b)<0 not satisfied!') %ceases execution
end
fa=f(a);
fb=f(b);
k=0;
while (b-a)/2>tol
  c=(a+b)/2;
  fc=f(c);
  if fc == 0                %c is a solution, done
    break
  end
  if sign(fc)*sign(fa)<0    %a and c make the new interval
    b=c;fb=fc;
  else                      %c and b make the new interval
    a=c;fa=fc;
  end
end
xc=(a+b)/2;                 %new midpoint is best estimate
```

为了调用 bisect.m, 首先定义一个 MATLAB 内联函数:

```
>> f=inline (x'x^3+x-1');
```

则命令

```
>> xc=bisect (f,0,1,0.00005)
```

返回的解精确到 0.000 05.

1.1.2 算法的精度和速度

如果 $[a,b]$ 是初始区间, 那么 n 次对分步骤之后, 区间 $[a_n,b_n]$ 的长度是 $(b-a)/2^n$. 选取中点 $x_c=(a_n+b_n)/2$ 作为对解 r 的最好估计, 它在真解的半个区间长度之内. 总结一下, 对分法进行 n 步之后, 我们求得

$$\text{解的误差} = |x_c - r| < \frac{b-a}{2^{n+1}}, \tag{1.1}$$

$$\text{计算函数值的次数} = n+2. \tag{1.2}$$

评估对分法效率的一种较好的方法是: 每计算一次函数值, 精度能提高多少. 每一步或者每计算一次函数值, 根的不确定度减少到原来的 1/2.

定义 1.3 *如果误差小于 0.5×10^{-p}, 那么解精确到 p 位小数.*

例 1.2 在区间 $[0,1]$ 中用对分法求 $f(x)=\cos x - x$ 的根, 精确到 6 位小数.

首先我们决定需要进行多少步对分法. 根据 (1.1), 在 n 步之后的误差是 $(b-a)/2^{n+1}=1/2^{n+1}$. 根据精确到 p 位小数的定义, 我们需要

$$\frac{1}{2^{n+1}} < 0.5\times10^{-6},$$
$$n > \frac{6}{\log_{10}2} \approx \frac{6}{0.301} = 19.9.$$

因此, 将需要 $n=20$ 步. 用对分法进行下去, 可以得到表 1-2. 精确到 6 位小数的近似解是 0.739 085. ◀

表 1-2

k	a_k	$f(a_k)$	c_k	$f(c_k)$	b_k	$f(b_k)$
0	0.000 000	+	0.500 000	+	1.000 000	−
1	0.500 000	+	0.750 000	−	1.000 000	−
2	0.500 000	+	0.625 000	+	0.750 000	−
3	0.625 000	+	0.687 500	+	0.750 000	−
4	0.687 500	+	0.718 750	+	0.750 000	−
5	0.718 750	+	0.734 375	+	0.750 000	−
6	0.734 375	+	0.742 188	−	0.750 000	−
7	0.734 375	+	0.738 281	+	0.742 188	−
8	0.738 281	+	0.740 234	−	0.742 188	−
9	0.738 281	+	0.739 258	−	0.740 234	−
10	0.738 281	+	0.738 770	+	0.739 258	−
11	0.738 769	+	0.739 014	+	0.739 258	−

(续)

k	a_k	$f(a_k)$	c_k	$f(c_k)$	b_k	$f(b_k)$
12	0.739 013	+	0.739 136	−	0.739 258	−
13	0.739 013	+	0.739 075	+	0.739 136	−
14	0.739 074	+	0.739 105	−	0.739 136	−
15	0.739 074	+	0.739 090	−	0.739 105	−
16	0.739 074	+	0.739 082	+	0.739 090	−
17	0.739 082	+	0.739 086	−	0.739 090	−
18	0.739 082	+	0.739 084	+	0.739 086	−
19	0.739 084	+	0.739 085	−	0.739 086	−
20	0.739 084	+	0.739 085	−	0.739 085	−

对于对分法来说, 运行多少步的问题是一个简单问题——只要像 (1.1) 那样选取所要的精度, 然后就能求出必需的步骤数. 我们将看到更大的算法往往不好预测, 而且也不同于 (1.1). 在那些情形中, 我们将建立明确的 "停止准则", 来控制算法终止的条件. 甚至对于对分法, 计算机运算的有限精度也会对可能正确的数字个数给出限制. 我们将在 1.3 节进一步探讨这个问题.

习题 1.1

1. 用介值定理求长度为 1 的区间, 使它包含以下方程的根:

(a) $x^3 = 9$; **(b)** $3x^3 + x^2 = x + 5$; **(c)** $\cos^2 x + 6 = x$.

2. 用介值定理求长度为 1 的区间, 使它包含以下方程的根:

(a) $x^5 + x = 1$; **(b)** $\sin x = 6x + 5$; **(c)** $\ln x + x^2 = 3$.

3. 考虑习题 1 中的方程, 执行两步对分法, 求一个真解的 $\frac{1}{8}$ 以内的近似根.

4. 考虑习题 2 中的方程, 执行两步对分法, 求一个真解的 $\frac{1}{8}$ 以内的近似根.

5. 考虑方程 $x^4 = x^3 + 10$.

(a) 求一个长度为 1 的区间 $[a, b]$, 使方程在其中有一个解.

(b) 以 $[a, b]$ 开始, 要算出在 10^{-10} 之内的解需要用多少步对分法? 用整数回答.

6. 假定初始区间为 $[-2, 1]$, 用对分法求函数 $f(x) = \frac{1}{x}$ 的根, 此方法收敛于实数吗? 它是根吗?

计算机问题 1.1

1. 用对分法求根, 精确到 6 位小数.

(a) $x^3 = 9$; **(b)** $3x^3 + x^2 = x + 5$; **(c)** $\cos^2 x + 6 = x$.

2. 用对分法求根, 精确到 8 位小数.

(a) $x^5 + x = 1$; **(b)** $\sin x = 6x + 5$; **(c)** $\ln x + x^2 = 3$.

3. 用对分法求以下方程的所有解. 用 MATLAB 中的`plot`命令来画出函数的图像, 并且确定包含根的长度为 1 的 3 个区间. 然后求根, 精确到 6 位小数.

(a) $2x^3 - 6x - 1 = 0$; **(b)** $e^{x-2} + x^3 - x = 0$; **(c)** $1 + 5x - 6x^3 - e^{2x} = 0$.

4. 用对分法解 $x^2 - A = 0$, 计算以下数的平方根, 精确到 8 位精确小数. 这里的 A 是 (a)2,

(b)3, (c)5. 指出你的初始区间以及所需要的步数.

5. 用对分法解 $x^3 - A = 0$, 计算以下数的立方根, 精确到 8 位精确小数. 这里的 A 是 (a)2, (b)3, (c)5. 指出你的初始区间以及所需要的步数.
6. 用对分法计算 $\cos x = \sin x$ 在区间 $[0,1]$ 中的解, 精确到 6 位小数.
7. 用对分法求精确到 6 位小数的两个实数 x, 它们使得矩阵

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & x \\ 4 & 5 & x & 6 \\ 7 & x & 8 & 9 \\ x & 10 & 11 & 12 \end{bmatrix}$$

的行列式等于 1 000. 对于你求出的每一个解, 计算相应的行列式并进行检查, 报告当用你的解时, 这个行列式有多少位精确小数 (小数点后面)(在 1.2 节中, 我们称它为与近似解相关的 "后向误差". 可以用 MATLAB 的`det`命令来计算行列式.

8. **Hilbert矩阵**是 $n \times n$ 矩阵, 其中第 i 行第 j 列元素是 $1/(i+j-1)$. 设 $\boldsymbol{A}$ 表示 5×5 阶的 Hilbert 矩阵, 它的最大特征值大约是 1.567. 用对分法确定怎样改变左上角元素 A_{11} 使 $\boldsymbol{A}$ 的最大特征值等于 π. A_{11} 的值要求精确到 6 位小数. 可以用 MATLAB 命令`hilb`、`pi`、`eig`和`max`简化你的工作.
9. 把 1 立方米的水放在半径为 1 米的球形罐里, 求水能达到的高度. 你的答案精度要求为 ±1 毫米. (提示: 注意到球罐中的水不到一半. 若球罐半径为 R, 水面到球底部高度为 H, 则水的体积是 $\pi H^2(R - 1/3H)$.)

1.2 不动点迭代

用计算器或计算机对任一个初始数重复应用余弦函数. 如果你用计算器, 确保是在弧度模式下进行的. 继续上面的步骤直到数字不再改变. 结果得到的数列至少前 10 位小数将收敛到 0.739 085 133 2. 本节的目的是解释这种计算 (不动点迭代 (FPI) 的一个例子) 为什么会收敛. 随后也将讨论算法收敛性的主要问题.

1.2.1 函数的不动点

通过迭代计算余弦函数所产生的数列收敛于数 r. 余弦的连续使用并不改变这个数. 对这个数, 余弦函数的输出等于输入, 或者说 $\cos r = r$.

定义 1.4 *如果 $g(r) = r$, 那么实数 r 是函数 g 的一个***不动点**(fixed point).

数 $r = 0.739\ 085\ 133\ 2$ 是函数 $g(x) = \cos x$ 的一个近似不动点. 函数 $g(x) = x^3$ 有 3 个不动点: $r = -1, 0, 1$.

我们用例 1.2 中的对分法去解方程 $\cos x - x = 0$. 不动点方程 $\cos x = x$ 是从不同的视角提出的同一问题. 当输出等于输入时, 那个数便是 $\cos x$ 的一个不动点, 同时也是方程 $\cos x - x = 0$ 的一个解.

一旦方程写成 $g(x)=x$, 通过初始估计 (initial guess)x_0 开始迭代计算函数 g, 不动点迭代就开始了.

不动点迭代

$$x_0 = \text{初始估计},$$
$$x_{i+1} = g(x_i), \qquad i = 0, 1, 2, \cdots$$

由此可知:

$$x_1 = g(x_0),$$
$$x_2 = g(x_1),$$
$$x_3 = g(x_2),$$
$$\vdots$$

当步数趋于无穷时, 数列 x_i 可能收敛, 也可能不收敛. 然而, 如果 g 连续而且 x_i 收敛 (譬如收敛到数 r), 那么 r 就是一个不动点. 事实上, 定理 0.5 意味着

$$g(r) = g\left(\lim_{i\to\infty} x_i\right) = \lim_{i\to\infty} g(x_i) = \lim_{i\to\infty} x_{i+1} = r. \tag{1.3}$$

用于内联函数 g 的不动点迭代算法容易写成以下 MATLAB 代码:

```
%Program 1.2 Fixed-Point Iteration
%Computes approximate solution of g(x)=x
%Input: inline function g, starting guess x0,
%       number of iteration steps k
%Output: Approximate solution xc
function xc=fpi(g, x0, k)
x(1)=x0;
for i=1:k
  x(i+1)=g(x(i));
end
x'                     %transpose output to a column
xc=x(k+1);
```

每一个方程 $f(x)=0$ 都能转化成一个不动点问题 $g(x)=x$ 吗? 是的, 而且有许多不同的方式. 例如, 在例 1.1 中的求根方程

$$x^3 + x - 1 = 0 \tag{1.4}$$

能写成

$$x = 1 - x^3, \tag{1.5}$$

因此, 我们可以定义 $g(x) = 1 - x^3$. 另外, (1.4) 中的 x^3 这一项能单独列出来, 得到

$$x = \sqrt[3]{1-x}, \tag{1.6}$$

这里的 $g(x) = \sqrt[3]{1-x}$. 第 3 种方法并不是很明显, 我们可以在 (1.4) 两边加上 $2x^3$, 得到

$$\begin{aligned} 3x^3 + x &= 1 + 2x^3, \\ (3x^2+1)x &= 1 + 2x^3, \\ x &= \frac{1+2x^3}{1+3x^2}, \end{aligned} \tag{1.7}$$

因此, 可以定义 $g(x) = (1+2x^3)/(1+3x^2)$.

下面对前面选取的 3 种 $g(x)$ 来示范不动点迭代. 其实我们求解的方程是 $x^3 + x - 1 = 0$. 首先, 考虑形式 $x = g(x) = 1 - x^3$, 初始点 $x_0 = 0.5$ 是任选的. 应用 FPI 得到表 1-3.

表 1-3

i	x_i	i	x_i
0	0.500 000 00	7	0.999 999 96
1	0.875 000 00	8	0.000 000 12
2	0.330 078 13	9	1.000 000 00
3	0.964 037 47	10	0.000 000 00
4	0.104 054 19	11	1.000 000 00
5	0.998 873 38	12	0.000 000 00
6	0.003 376 06		

迭代并不收敛而是交替地趋于 0 和 1 这两个数. 因为 $g(0) = 1$, $g(1) = 0$, 所以这两个数都不是不动点. 不动点迭代失败. 用对分法我们知道, 如果 f 在原区间连续, 而且 $f(a)f(b) < 0$, 则必收敛于根, 但对于 FPI 并不是如此.

第二种选取是 $g(x) = \sqrt[3]{1-x}$, 我们将使用相同的初始估计 $x_0 = 0.5$, 见表 1-4.

表 1-4

i	x_i	i	x_i
0	0.500 000 00	13	0.684 544 01
1	0.793 700 53	14	0.680 737 37
2	0.590 880 11	15	0.683 464 60
3	0.742 363 93	16	0.681 512 92
4	0.636 310 20	17	0.682 910 73
5	0.713 800 81	18	0.681 910 19
6	0.659 006 15	19	0.682 626 67
7	0.698 632 61	20	0.682 113 76
8	0.670 448 50	21	0.682 481 02
9	0.690 729 12	22	0.682 218 09
10	0.676 258 92	23	0.682 406 35
11	0.686 645 54	24	0.682 271 57
12	0.679 222 34	25	0.682 368 07

这一次 FPI 成功了. 这种迭代显然收敛于接近 0.682 3 的一个数.

最后我们用重新整理的 $x = g(x) = (1+2x^3)/(1+3x^2)$. 像前一种情形一样, 它是收敛的, 而且收敛得更显著, 见表 1-5.

表 1-5

i	x_i	i	x_i
0	0.500 000 00	4	0.682 327 80
1	0.714 285 71	5	0.682 327 80
2	0.683 179 72	6	0.682 327 80
3	0.682 328 42	7	0.682 327 80

经过 4 步不动点迭代后, 我们已经有 4 位精确的数字, 而且后面很快会有更多精确的数字. 与前面的尝试相比, 这是一个惊人的结果. 我们的下一个目标是试图解释这 3 种结果的差别.

1.2.2 不动点迭代的几何原理

在 1.2.1 节, 我们发现把方程 $x^3 + x - 1 = 0$ 改写成不动点问题的 3 种不同方法的结果显然不同. 要弄清楚为什么 FPI 方法在有些场合收敛而在另一些场合不收敛, 考虑该方法的几何形状将会有帮助.

图 1-3 显示了以前讨论过的 3 种不同的 $g(x)$, 以及每种情况下最初的几步 FPI. 对每一个 $g(x)$, 不动点 r 是相同的. 用曲线 $y = g(x)$ 和 $y = x$ 的交点表示它. FPI 的每一步可以通过画出两条线段作为示意图: (1) **垂直方向交于曲线**, (2) **水平方向交于对角线**$y = x$. 图 1-3 中垂直和水平箭头对应着 FPI 的每一步. 垂直箭头从 x 值指向函数 g, 表示 $x_i \to g(x_i)$. 水平箭头表示把对应于 y 轴上的输出 $g(x_i)$ 转换成在 x 轴上的相同的数 x_{i+1}, 准备在下一步输入 g. 这是通过从输出高度 $g(x_i)$ 向对角线 $y = x$ 画水平线段而得. 此不动点迭代的几何图形称为**蛛网图**(cobweb diagram).

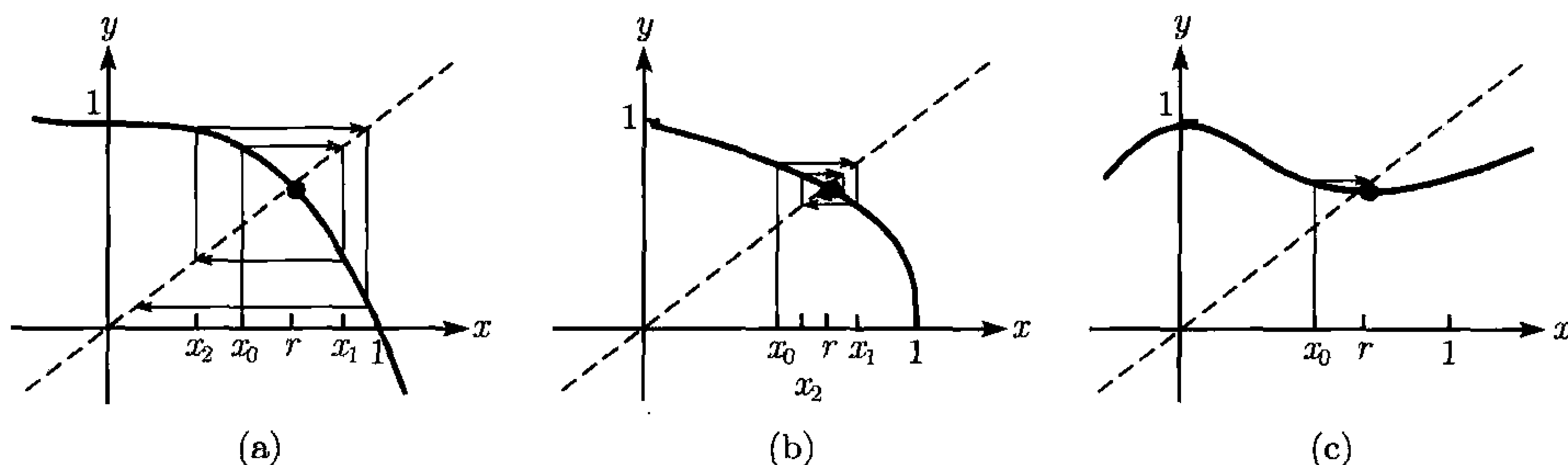


图 1-3 FPI 的几何思想. 不动点是 $g(x)$ 的和对角线的交点, $g(x)$ 的 3 种形式及 FPI 的前几步如图所示: (a) $g(x) = 1 - x^3$; (b) $g(x) = (1-x)^{\frac{1}{3}}$; (c) $g(x) = (1+2x^3)/(1+3x^2)$

在图 1-3a 中, 路径从 $x_0 = 0.5$ 出发, 向上移动到函数, 然后水平地移到对角

线上的点 (0.875, 0.875), 这就是 (x_1, x_1). 下一步应把 x_1 代入 $g(x)$. 和对 x_0 所做的一样, 垂直地移动到函数. 这样便得到 $x_2 \approx 0.330\,0$, 而且通过水平移动把 y 值移动到 x 值后, 我们继续使用相同的方法得到 $x_3, x_4, \cdots$. 像我们以前看到的那样, 对这个 $g(x)$, FPI 并不成功. 迭代最终交替地趋于 0 和 1, 它们都不是不动点.

不动点迭代在图 1-3b 中比较成功. 尽管这里的 $g(x)$ 粗看与 (a) 部分的 $g(x)$ 相似, 却有着重要的差别. 这一点我们将在 1.2.3 节中阐明. 你可能想推测差别在哪里. 是什么使得 FPI 盘旋地指向 (b) 中的不动点, 却盘旋地离开在 (a) 中的不动点? 图 1-3c 表示一个快速收敛的例子. 这张图有助于你的推测吗? 如果你猜测靠近不动点处 $g(x)$ 的斜率起了某种作用, 那么你是正确的.

1.2.3 不动点迭代的线性收敛性

通过仔细考察在最简单情形下的算法就很容易解释 FPI 的收敛性质. 图 1-4 表示两个线性函数 $g_1(x) = -\frac{3}{2}x + \frac{5}{2}$ 和 $g_2(x) = -\frac{1}{2}x + \frac{3}{2}$ 的不动点迭代. 在每一种情形中, 不动点 $x = 1$, 但是 $|g_1'(1)| = |-\frac{3}{2}| > 1$, 而 $|g_2'(1)| = |-\frac{1}{2}| < 1$, 按照描述 FPI 的垂直和水平箭头, 我们明白了差别的原因. 因为 g_1 在不动点的斜率大于 1, 所以代表从 x_n 变化到 x_{n+1} 的那些垂直线段, 当 FPI 进行时, 长度会增加. 结果, 即使初始估计 x_0 十分靠近不动点, 迭代也会 "盘旋离开" 不动点 $x = 1$. g_2 的情形相反: g_2 的斜率小于 1, 垂直线段在长度上减小, 因此 FPI"盘旋朝向" 不动点. 因此, 正是 $|g'(r)|$ 决定了发散还是收敛.

这是几何思想. 就方程而言, 用 $x - r$ 来写 $g_1(x)$ 和 $g_2(x)$ 会对我们有所帮助, 这里 $r = 1$ 是不动点:

$$
\begin{aligned}
g_1(x) &= -\frac{3}{2}(x-1) + 1, \\
g_1(x) - 1 &= -\frac{3}{2}(x-1), \\
x_{i+1} - 1 &= -\frac{3}{2}(x_i - 1).
\end{aligned} \tag{1.8}
$$

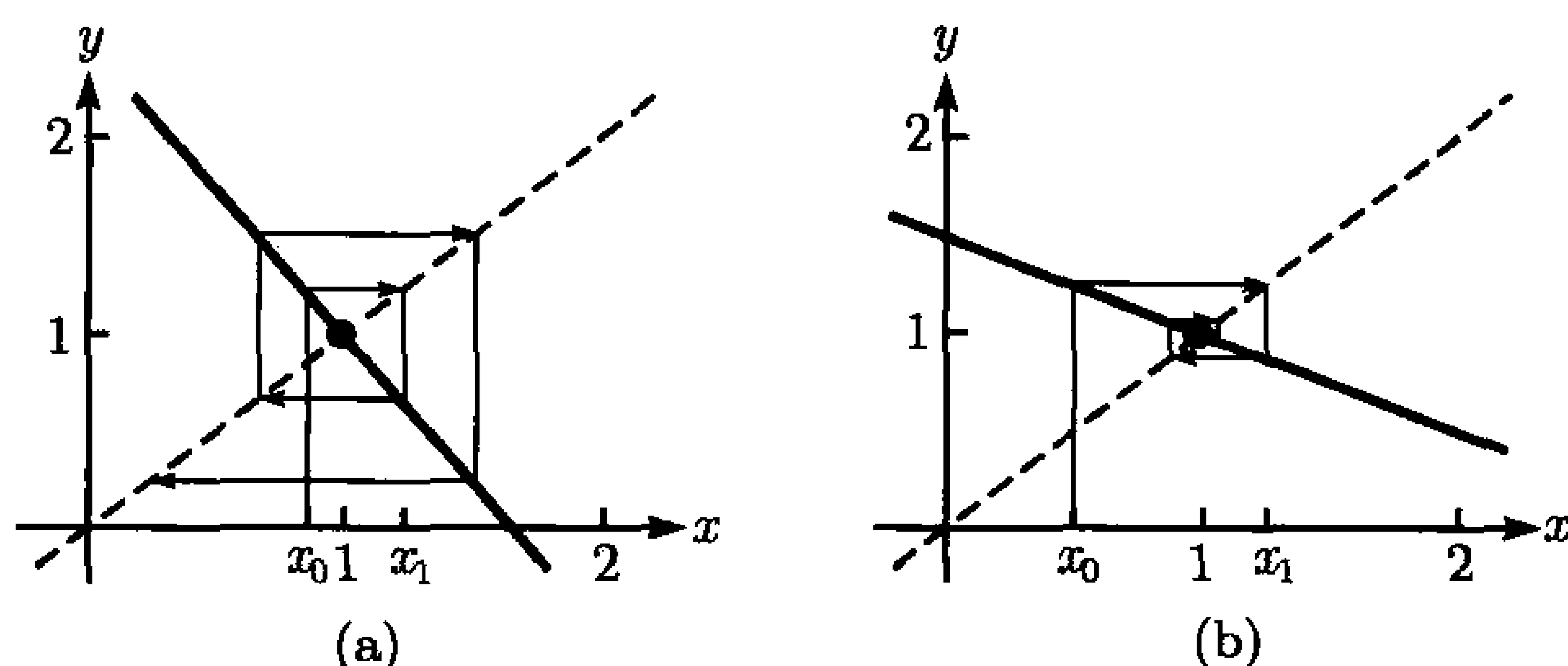


图 1-4 线性函数的蛛网图. (a) 如果线性函数的斜率绝对值大于 1, 则当 FPI 进行时, 附近的估计点会远离不动点, 导致该方法失败. (b) 斜率的绝对值小于 1 时, 情形正好相反, 因此求得不动点

如果把 $e_i = |r - x_i|$ 看作在第 i 步的误差 (即在第 i 步时最佳估计点到不动点的距离), 我们从 (1.8) 看到, $e_{i+1} = 3e_i/2$, 它表示误差在每一步以近似于 3/2 的因子而增大, 因此这是发散的.

对 g_2 重复前面的代数运算, 我们有

$$\begin{aligned} g_2(x) &= -\frac{1}{2}(x-1)+1, \\ g_2(x)-1 &= -\frac{1}{2}(x-1), \\ x_{i+1}-1 &= -\frac{1}{2}(x_i-1). \end{aligned}$$

结果是 $e_{i+1} = e_i/2$, 意味着误差 (即到不动点的距离) 在每一步乘以 $\frac{1}{2}$. 当步数增加时, 误差减少到零. 这是一种特殊类型的收敛性.

定义 1.5 *设 e_i 表示迭代法在第 i 步的误差, 如果*

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = S < 1,$$

*则称这种方法满足速度是 S 的***线性收敛**(linear convergence).

不动点迭代对 g_2 来说是以速度 $S = \frac{1}{2}$ 线性收敛于根 $r = 1$. 因为 g_1 和 g_2 是线性的, 所以前面的讨论被简化了, 但是如下面定理所示, 同样的推理适用于具有不动点 $g(r) = r$ 的一般的连续可微函数 $g(x)$.

定理 1.6 *假设 g 是连续可微函数, 满足 $g(r) = r$ 及 $S = |g'(r)| < 1$, 那么对充分接近 r 的初始估计, 不动点迭代以速度 S 线性地收敛于不动点 r.*

证 *设 x_i 表示第 i 步迭代. 根据中值定理, 在 x_i 和 r 之间存在常数 c_i, 使得*

$$x_{i+1} - r = g'(c_i)(x_i - r), \tag{1.9}$$

这里我们已代入 $x_{i+1} = g(x_i)$ 及 $r = g(r)$. 定义 $e_i = |x_i - r|$, (1.9) 就能写成

$$e_{i+1} = |g'(c_i)|e_i. \tag{1.10}$$

如果 $S = |g'(r)|$ 小于 1, 那么根据 g' 的连续性, 存在 r 的一个小邻域, 在其中有 $|g'(x)| < (S+1)/2$. 它略大于 S 但仍小于 1. 如果 x_i 碰巧在这个邻域内, 那么 c_i 也在此邻域内 (它被夹在 x_i 和 r 之间), 所以

$$e_{i+1} \leqslant \frac{S+1}{2} e_i.$$

于是误差将在这一步和以后的每一步都以因子 $(S+1)/2$ 或者更佳的因子减小. 这就意味着 $\lim\limits_{i\to\infty} x_i = r$, 因此对 (1.10) 取极限得到

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = \lim_{i\to\infty} |g'(c_i)| = |g'(r)| = S.$$

根据定理 1.6, 近似误差关系

$$e_{i+1} \approx S e_i \tag{1.11}$$

在逼近收敛的极限中成立, 这里 $S = |g'(r)|$. 对这个定理的变形见习题 15.

定义 1.7 *如果对充分接近 r 的初始估计来说迭代方法收敛, 那么该方法称为***局部收敛***到 r.*

换言之, 如果存在邻域 $(r-\varepsilon, r+\varepsilon)$, 这里 $\varepsilon > 0$, 使得从该邻域内的任意初始估计出发, 该方法都收敛到 r, 则这种方法局部收敛到 r. 定理 1.6 的结论是: 如果 $|g'(r)| < 1$, 那么不动点迭代局部收敛.

定理 1.6 解释了在前面的 $f(x) = x^3 + x - 1 = 0$ 的不动点迭代中所发生的事情. 我们知道, 根 $r \approx 0.682\,3$. 对于 $g(x) = 1 - x^3$, 导数是 $g'(x) = -3x^2$. 在根 r 附近, FPI 相当于 $e_{i+1} \approx S e_i$, 这里的 $S = |g'(r)| = |-3(0.682\,3)^2| \approx 1.396\,6 > 1$, 所以误差增大, 因此不可能收敛. 这种在 e_{i+1} 和 e_i 之间的误差关系仅在 r 附近成立, 但它确实表示对 r 可能出现不收敛性.

对于第二种选择, $g(x) = \sqrt[3]{1-x}$, 导数 $g'(x) = \frac{1}{3}(1-x)^{\frac{-2}{3}}(-1)$, 以及 $K = |(1-0.682\,3)^{-\frac{2}{3}}/3| \approx 0.716 < 1$. 定理 16 意味着收敛, 与我们前面的计算一致.

对于第三种选择, $g(x) = (1+2x^3)/(1+3x^2)$, 导数

$$g'(x) = \frac{6x^2(1+3x^2) - (1+2x^3)6x}{(1+3x^2)^2} = \frac{6x(x^3+x-1)}{(1+3x^2)^2},$$

因此 $S = |g'(r)| = 0$, 这是能取到的最小的 S, 这就导致了图 1-3c 中显示的非常快的收敛.

例 1.3 解释不动点迭代 $g(x) = \cos x$ 为什么收敛.

这是本章前面许诺过的解释. 反复应用余弦函数, 对应于 $g(x) = \cos x$ 的 FPI. 根据定理 1.6, 解为 $r \approx 0.74$, 因为 $g'(r) = -\sin r \approx -\sin 0.74 \approx -0.67$ 的绝对值小于 1, 所以解 $r \approx 0.74$ 附近的估计点收敛于它. ◀

例 1.4 用不动点迭代求 $\cos x = \sin x$ 的根.

把方程转化成不动点问题的最简方法是在方程的两边加上 x. 可以把问题重写为

$$x + \cos x - \sin x = x,$$

并定义

$$g(x) = x + \cos x - \sin x. \tag{1.12}$$

对于这个 $g(x)$, 不动点迭代的结果如表 1-6 所示.

表 1-6

i	x_i	$g(x_i)$	$e_i = \|x_i - r\|$	e_i/e_{i-1}
0	0.000 000 0	1.000 000 0	0.785 398 2	
1	1.000 000 0	0.698 831 3	0.214 601 8	0.273
2	0.698 831 3	0.821 102 5	0.086 566 9	0.403
3	0.821 102 5	0.770 619 7	0.035 704 3	0.412
4	0.770 619 7	0.791 518 9	0.014 778 5	0.414
5	0.791 518 9	0.782 862 9	0.006 120 7	0.414
6	0.782 862 9	0.786 448 3	0.002 535 3	0.414
7	0.786 448 3	0.784 963 2	0.001 050 1	0.414
8	0.784 963 2	0.785 578 3	0.000 435 0	0.414
9	0.785 578 3	0.785 323 5	0.000 180 1	0.414
10	0.785 323 5	0.785 429 1	0.000 074 7	0.415
11	0.785 429 1	0.785 385 4	0.000 030 9	0.414
12	0.785 385 4	0.785 403 5	0.000 012 8	0.414
13	0.785 403 5	0.785 396 0	0.000 005 3	0.414
14	0.785 396 0	0.785 399 1	0.000 002 2	0.415
15	0.785 399 1	0.785 397 8	0.000 000 9	0.409
16	0.785 397 8	0.785 398 3	0.000 000 4	0.444
17	0.785 398 3	0.785 398 1	0.000 000 1	0.250
18	0.785 398 1	0.785 398 2	0.000 000 1	1.000
19	0.785 398 2	0.785 398 2	0.000 000 0	

在这张表中有几处地方值得关注. 首先这个迭代看起来收敛于 0.785 398 2. 因为

$$\cos\frac{\pi}{4} = \frac{\sqrt{2}}{2} = \sin\frac{\pi}{4},$$

所以方程 $\cos x - \sin x = 0$ 的真解是 $r = \frac{\pi}{4} \approx 0.785\,398\,2$. 第 4 列是 "误差列". 它表示在第 i 步, 最佳估计 x_i 和实际不动点 r 之差的绝对值. 在靠近表的底部这个差变小, 说明估计点向不动点收敛.

注意误差列中的模式. 误差似乎以常数因子递减, 每个误差比前一误差的一半还稍微小一点. 更精确地说, 相继两个误差的比由最后一列所示. 我们在表的绝大部分看到, 相继两个误差的比 e_{i+1}/e_i 趋于常数, 大约是 0.414. 换言之, 我们看到线性收敛关系

$$e_i \approx 0.414 e_{i-1}. \tag{1.13}$$

因为定理 1.6 蕴含着

$$S = |g'(r)| = |1 - \sin r - \cos r| = \left|1 - \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}\right| = |1 - \sqrt{2}| \approx 0.414,$$

所以这恰好符合我们的预期. ◀

细心的读者会注意到表末尾的一个矛盾. 在计算误差 e_i 中, 我们对准确的不动点 r 仅用了 7 位有效数字. 结果, 当 e_i 接近 10^{-8} 时 e_i 的相对精度比较差, 比值 e_i/e_{i-1} 变得不精确. 如果我们对 r 用一个精确得多的值, 这一问题将不再出现.

例 1.5 求 $g(x) = 2.8x - x^2$ 的不动点.

函数 $g(x) = 2.8x - x^2$ 有两个不动点 0 和 1.8, 这可以通过手工解 $g(x) = x$ 来确定, 或者在图 1-5 中通过 $y = g(x)$ 和 $y = x$ 的曲线相交来确定. 图 1-5 表示取初始估计 $x = 0.1$ 的 FPI 蛛网图. 对于这个例子, 迭代

$$x_0 = 0.100\,0, \quad x_1 = 0.270\,0, \quad x_2 = 0.683\,1, \quad x_3 = 1.446\,1, \quad x_4 = 1.957\,9,$$

如此下去, 可以看出交点是沿着对角线的.

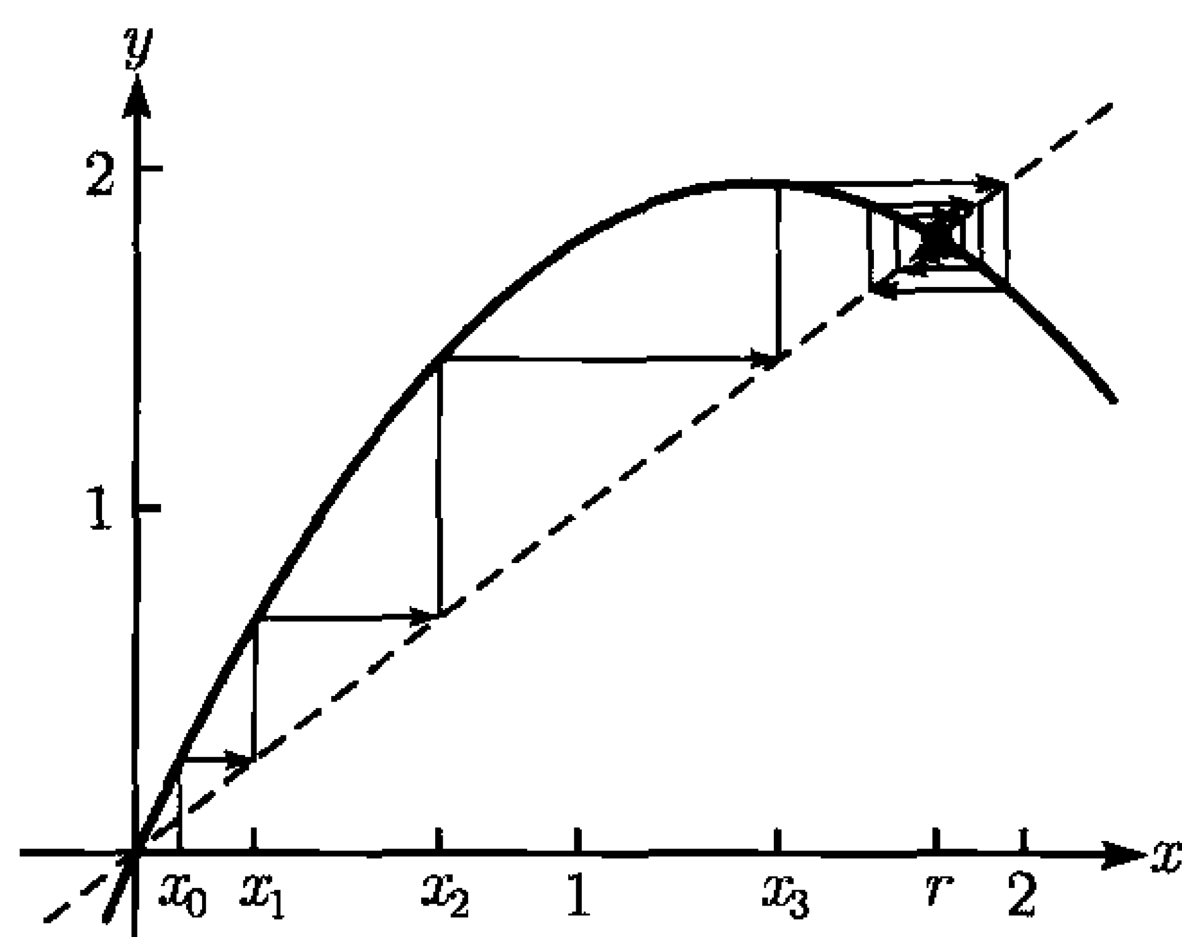


图 1-5 不动点迭代的蛛网图. 例 1.5 有两个不动点: 0 和 1.8. 图中所示的是初始估计为 0.1 的迭代, 用 FPI 方法仅收敛到 1.8

尽管初始点 $x_0 = 0.1$ 靠近不动点 0, 但是 FPI 朝另一个不动点 $x = 1.8$ 移动, 并且收敛到那里. 两个不动点之间的区别是 g 在 $x = 1.8$ 处的斜率, 由 $g'(1.8) = -0.8$ 给出, 其绝对值小于 1. 另一方面, g 在另一个不动点 $x = 0$(排斥点) 处的斜率是 $g'(0) = 2.8$, 其绝对值大于 1.

定理 1.6 在*后验估计*(posteriori) 中相当有用——在 FPI 计算的结尾, 我们求得根并且能够一步一步地计算误差. 这个定理从侧面说明为什么收敛速度 S 有上述表现. 在计算开始之前就获知这一信息可能更为有用. 下一个例子表明, 在某些情形下, 我们确实能够这样做. ◄

例 1.6 用 FPI 计算 $\sqrt{2}$.

古代求平方根的方法可以表示为一种 FPI. 假设我们想求 $\sqrt{2}$ 的前 10 位数字. 从初始估计 $x_0 = 1$ 开始. 这个估计值明显太小, 而 $\frac{2}{1} = 2$ 又太大. 事实上, 任一初始估计 $0 < x_0 < 2$ 与 $\frac{2}{x_0}$ 一起会将 $\sqrt{2}$ 括在其中. 因此, 把这两个数平均后得到更好的估计:

$$x_1 = \frac{1 + \frac{2}{1}}{2} = \frac{3}{2}.$$

现在再重复一次. 尽管 $\frac{3}{2}$ 接近了一些, 但它比 $\sqrt{2}$ 大, 而 $\frac{2}{\frac{3}{2}}=\frac{4}{3}$ 却太小. 如前, 平均后得到

$$x_2=\frac{\frac{3}{2}+\frac{4}{3}}{2}=\frac{17}{12}=1.41\bar{6},$$

它更接近 $\sqrt{2}$. 再一次, x_2 和 $\frac{2}{x_2}$ 把 $\sqrt{2}$ 括在其中.

下一步得到

$$x_3=\frac{\frac{17}{12}+\frac{24}{17}}{2}=\frac{577}{408}\approx 1.414\ 215\ 686.$$

用计算器检查可以看到, 这个估计与 $\sqrt{2}$ 仅相差 3×10^{-6}. 我们正在实施的 FPI 是

$$x_{i+1}=\frac{x_i+\frac{2}{x_i}}{2}. \tag{1.14}$$

注意到 $\sqrt{2}$ 正是这个迭代的不动点.

在结束计算之前, 我们来判断它是否收敛. 根据定理 6.1, 我们需要 $S<1$. 对这个迭代, $g(x)=\frac{1}{2}(x+\frac{2}{x})$, $g'(x)=\frac{1}{2}(1-\frac{2}{x^2})$, 在不动点处计算得到

$$g'(\sqrt{2})=\frac{1}{2}\left(1-\frac{2}{(\sqrt{2})^2}\right)=0, \tag{1.15}$$

所以 $S=0$, 我们断定: 这个迭代将收敛并且收敛得非常快.

习题 1.2 节中第 8 题的问题是: 求任一正数的平方根时, 这种方法是否一定成功? ◄

亮点　收敛性

例 1.6 的巧妙方法仅在 3 步后就收敛到 $\sqrt{2}$ 的 5 位小数. 这种简单方法是数学史中最古老的一种. 图 1-6a 所示的楔形片 YBC7289 是 1962 年在巴格达附近发现的, 大约起源于公元前 1750 年. 它包括了表示面积为 2 的正方形边长的基为 60 的近似数 (1)(24)(51)(10). 以 10 为基, 它就是

$$1+\frac{24}{60}+\frac{51}{60^2}+\frac{10}{60^3}=1.414\ 212\ 96.$$

我们不知道巴比伦人的算法, 但是有人推测, 他们正是采用了例 1.6 的算法, 但是以习惯的 60 为基. 无论如何, 这种方法出现在由亚历山大人 Heron(海伦) 写于公元 1 世纪的 *Metrica*(第 1 册) 中, 用来计算 $\sqrt{720}$.

1.2.4 停止准则

与对分法的情形不同, 对于 FPI 收敛到给定的偏差之内所需要的步数很少可以事先预测. 在没有类似 (1.1) 那种对分法的误差公式时, 我们必须作出终止算法

的决定, 称它为**停止准则**(stopping criterion).

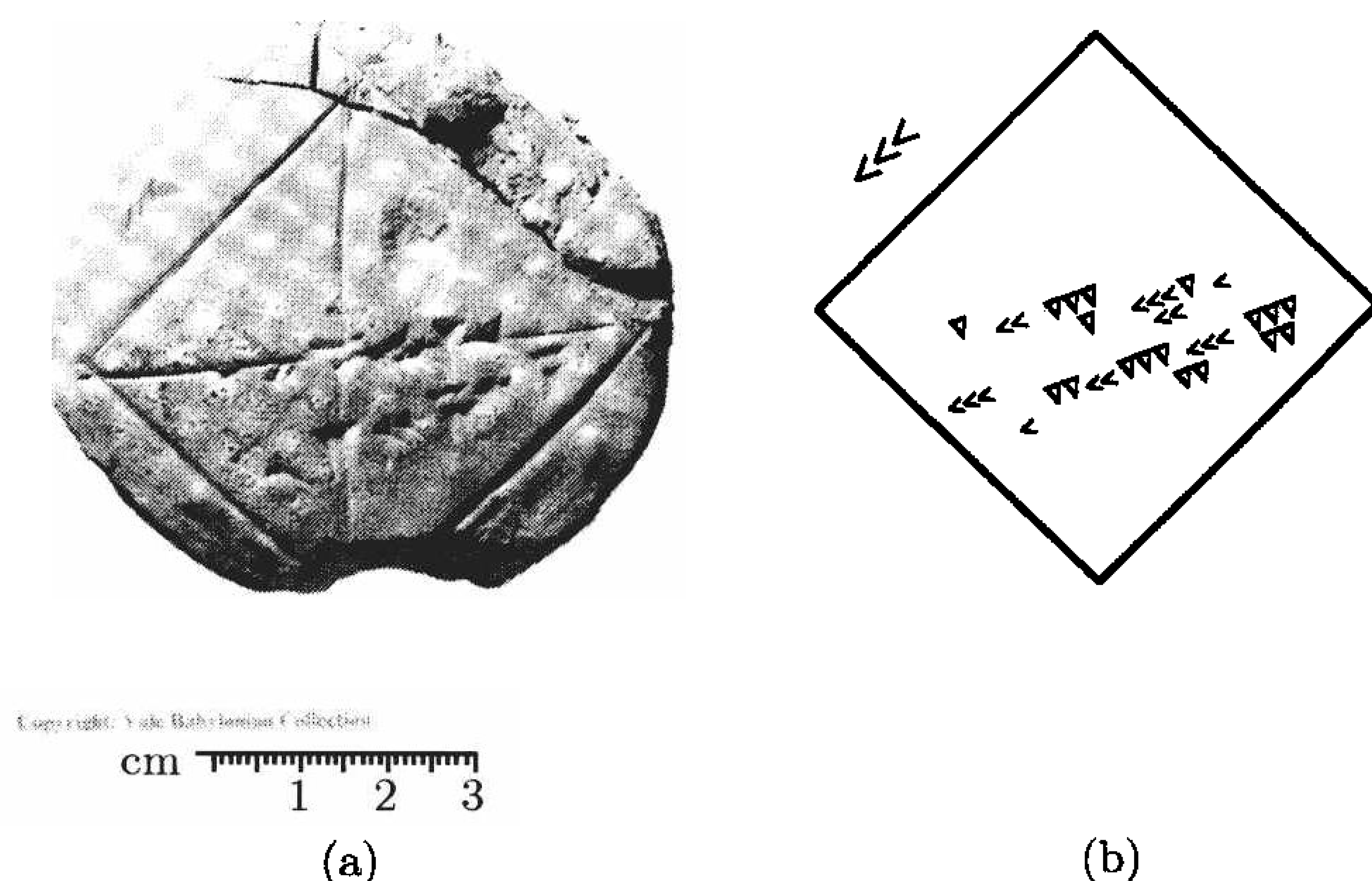


图 1-6 $\sqrt{2}$ 的古代计算: (a) 楔形片 YBC7289, (b) 楔形片的示意图. 巴比伦人以基 60 计算, 但使用基 10 的记法. "<" 表示 10, "∇" 表示 1. 在左上方是 30, 即边的长度, 沿着中间是 1, 24, 51 和 10, 它们表示精确到 5 位小数的 2 的平方根, 在下方的数 42, 25, 35 表示基为 60 时的 $30\sqrt{2}$

对一组容限 TOL, 可以要求绝对误差停止准则

$$|x_{i+1} - x_i| < \text{TOL}, \tag{1.16}$$

或者在解不是太接近于零的情形, 可以要求相对误差停止准则

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < \text{TOL}. \tag{1.17}$$

混合的绝对/相对停止准则

$$\frac{|x_{i+1} - x_i|}{\max(|x_{i+1}|, \theta)} < \text{TOL} \tag{1.18}$$

在解接近于零的情况下, 对于某个 $\theta > 0$ 将非常有用. 另外, 好的 FPI 代码为了防止收敛失败, 对最大迭代步数作了限制. 停止准则的讨论相当重要, 当我们在 1.3 节学习前向和后向误差时将再次讨论它, 并采用更复杂的方式.

对分法保证线性收敛. 不动点迭代仅是局部收敛, 而当它收敛时它是线性收敛. 这两种方法要求每一步计算一次函数值. 对分法每进行一步, 不确定度就减少 $\frac{1}{2}$. 与之相比, FPI 每进行一步, 不确定度近似减少 $S = |g'(r)|$. 因此, 不动点迭代可能比对分法快也可能比它慢, 这依赖于 S 是小于还是大于 $\frac{1}{2}$. 1.4 节将介绍 FPI 的一种特别精细的形式, 即 Newton 方法, 这时 S 为零.

习题 1.2

1. 用定理 1.6 确定 $g(x)$ 的不动点迭代是否局部收敛于给定的不动点 r.

   (a) $g(x)=(2x-1)^{\frac{1}{3}}, r=1$;　(b) $g(x)=(x^3+1)/2, r=1$;　(c) $g(x)=\sin x+x, r=0$.

2. 用定理 1.6 确定 $g(x)$ 的不动点迭代是否局部收敛于给定的不动点 r.

   (a) $g(x)=(2x-1)/x^2, r=1$;　(b) $g(x)=\cos x+\pi+1, r=\pi$;　(c) $g(x)=\mathrm{e}^{2x}-1, r=0$.

3. 求每一个不动点并确定不动点迭代是否局部收敛于它.

   (a) $g(x)=\frac{1}{2}x^2+\frac{1}{2}x$;　(b) $g(x)=x^2-\frac{1}{4}x+\frac{3}{8}$.

4. 求每一个不动点并确定不动点迭代是否局部收敛于它.

   (a) $g(x)=x^2-\frac{3}{2}x+\frac{3}{2}$;　(b) $g(x)=x^2+\frac{1}{2}x-\frac{1}{2}$.

5. 用 3 种不同的方式把每个方程表示为不动点问题 $x=g(x)$.

   (a) $x^3-x+\mathrm{e}^x=0$;　(b) $3x^{-2}+9x^3=x^2$.

6. 考虑不动点迭代 $x\to g(x)=x^2-0.24$.

   (a) 你认为由不动点迭代计算根 -0.2, 譬如说大约到 10 位精确小数, 比对分法快还是慢?

   (b) 求出其他不动点, FPI 将收敛到它吗?

7. 验证 $\frac{1}{2}$ 和 -1 是 $f(x)=2x^2+x-1=0$ 的根. 把 x^2 项分离出来并求解 x, 来找出 $g(x)$ 的两个不动点. 哪一个根将通过不动点迭代求得?

8. 证明例 1.6 的方法可用于计算任一正数的平方根.

9. 对立方根考察例 1.6 的想法. 如果 x 是小于 $A^{\frac{1}{3}}$ 的估计, 那么 $\frac{A}{x^2}$ 将大于 $A^{\frac{1}{3}}$, 所以这两个数的平均将是比 x 更好的近似. 请提出基于这一事实的一种不动点迭代, 并用定理 1.6 确定它是否收敛于 A 的立方根.

10. 用加权平均改进习题 9 的立方根算法. 对某个固定的数 $0<w<1$, 令 $g(x)=wx+(1-w)\frac{A}{x^2}$, w 的最佳选择是什么?

11. 考虑关于 $g(x)=1-5x+\frac{15}{2}x^2-\frac{5}{2}x^3$ 的不动点迭代. (a) 证明 $1-\sqrt{\frac{3}{5}}, 1, 1+\sqrt{\frac{3}{5}}$ 是不动点. (b) 证明这 3 个不动点都不局部收敛 (计算机问题 7 进一步研究了这个例子).

12. 证明初始估计 0,1, 2 导出习题 11 中的不动点. 对于接近这些数的其他初始估计会发生什么现象?

13. 假设 $g(x)$ 连续可微并且不动点迭代 $g(x)$ 恰有 3 个不动点, $r_1<r_2<r_3$. 另假定 $|g'(r_1)|=0.5$ 及 $|g'(r_3)|=0.5$. 在这些条件下, 列出 $|g'(r_2)|$ 所有可能的值.

14. 假设 $g(x)$ 是连续可微函数, 并且不动点迭代 $g(x)$ 恰有 3 个不动点: $-3, 1, 2$. 另假定 $g'(-3)=2.4$ 以及从充分靠近不动点 2 处开始的 FPI 收敛于 2. 求 $g'(1)$.

15. 证明定理 1.6 的变形: 如果 g 在包含不动点 r 的区间 $[a,b]$ 上连续可微并且 $|g'(x)|\leqslant B<1$, 那么对于 $[a,b]$ 中的任一初始估计, FPI 均收敛于 r.

16. 证明在闭区间上满足 $|g'(x)|<1$ 的连续可微函数 $g(x)$ 在这个区间上不可能有两个不动点.

17. 考虑 $g(x)=x-x^3$ 的不动点迭代. (a) 证明 $x=0$ 是仅有的不动点. (b) 证明如果 $0<x_0<1$, 那么 $x_0>x_1>x_2>\cdots>0$. (c) 证明当 $g'(0)=1$ 时 FPI 收敛于 $r=0$. [提示: 运用每一个单调有界数列收敛 (于某个极限) 这一事实.]

18. 考虑 $g(x) = x + x^3$ 的不动点迭代. (a) 证明 $x = 0$ 是仅有的不动点. (b) 证明如果 $0 < x_0 < 1$, 那么 $x_0 < x_1 < x_2 < \cdots$. (c) 证明当 $g'(0) = 1$ 时, FPI 未能收敛于不动点. 与习题 17 一起说明当 $|g'(r)| = 1$ 时, FPI 可能收敛到不动点 r 也可能发散于 r.

19. 考虑方程 $x^3 + x - 2 = 0$ 有根 $r = 1$. 两边加上 cx 这一项并除以 c 得到 $g(x)$.
(a) 对什么样的 c, FPI 局部收敛于 $r = 1$? (b) 对什么样的 c, FPI 将收敛得最快?

20. 假定把不动点迭代用于二次连续可微函数 $g(x)$, 并且对不动点 $rg'(r) = 0$ 成立. 证明如果 FPI 收敛于 r, 那么误差满足 $\lim_{i\to\infty}(e_{i+1})/e_i^2 = M$, 这里 $M = |g''(r)|/2$.

21. 通过对方程 $x^2 + x = 5/16$ 分离 x 项来定义不动点迭代. 求出两个不动点, 并且确定哪个初始估计在 (不动点) 迭代下收敛于不动点 (提示: 画出 $g(x)$ 和蛛网图).

22. 求出所有使不动点迭代 $x \to \frac{4}{9} - x^2$ 收敛到不动点的初始估计的集合.

计算机问题 1.2

1. 用不动点迭代, 求以下方程的解, 精确到 8 位小数:
(a) $x^3 = 2x + 2$; (b) $\mathrm{e}^x + x = 7$; (c) $\mathrm{e}^x + \sin x = 4$.

2. 用不动点迭代, 求以下方程的解, 精确到 8 位小数:
(a) $x^5 + x = 1$; (b) $\sin x = 6x + 5$; (c) $\ln x + x^2 = 3$.

3. 用例 1.6 中的不动点迭代, 求以下各数的平方根, 精确到 8 位小数: (a)3, (b)5. 说出你的初始估计和所需要的步数.

4. 用 $g(x) = (2x + A/x^2)/3$ 的不动点迭代计算以下各数的立方根, 精确到 8 位小数, 这里的 A 是 (a)2, (b)3, (c)5. 说出你的初始估计和所需要的步数.

5. 例 1.3 表明 $g(x) = \cos x$ 是一个收敛的 FPI. 对 $g(x) = \cos^2 x$ 是否同样正确? 求其不动点, 精确到 6 位小数, 并报告所需要的 FPI 步数. 用定理 1.6 讨论局部收敛性.

6. 为了求以下方程的精确到 6 位小数的根, 请导出 3 种不同的 $g(x)$. 对每一种 $g(x)$ 运行 FPI, 并报告结果, 收敛还是发散. 每个方程 $f(x) = 0$ 有 3 个根. 如果需要的话, 导出更多的 $g(x)$, 直到用 FPI 求出所有的根. 对每一次收敛的迭代, 从误差 $\frac{e_{i+1}}{e_i}$ 确定 S 的值, 并与在 (1.11) 中计算所确定的 S 进行比较.
(a) $f(x) = 2x^3 - 6x - 1$; (b) $f(x) = \mathrm{e}^{x-2} + x^3 - x$; (c) $f(x) = 1 + 5x - 6x^3 - \mathrm{e}^{2x}$.

7. 习题 11 考虑了用于 $g(x) = 1 - 5x + \frac{15}{2}x^2 - \frac{5}{2}x^3 = x$ 的不动点迭代. 求以下 FPI 的初始估计: (a) 在区间 $(0, 1)$ 中无限循环; (b) 与 (a) 相同, 但区间是 $(1, 2)$; (c) 发散至无穷. 情形 (a) 和 (b) 是混沌动力学的例子. 在所有 3 种情形中, FPI 是不成功的.

1.3 精度的界限

数值分析的目标之一是按指定精度水平计算出答案. 在双精度下工作意味着我们进行存储和运算的数保持 52 位精度, 大约 16 位十进制数.

能永远把答案计算到 16 位正确的有效数字吗? 第 0 章曾经指出, 一个计算二次方程根的内生算法很可能损失一些或者全部有效数字. 改进的算法消除了这个问题. 在本节中, 我们将看到一些新的东西——双精度计算机的计算, 即使用最好

的算法也根本不可能接近 16 位精确数字.

1.3.1 前向误差和后向误差

第一个例子说明: 在某些情形下, 笔和纸仍可以胜过计算机.

例 1.7 用对分法求 $f(x)=x^3-2x^2+\frac{4}{3}x-\frac{8}{27}$ 的根, 精确到 6 位有效数字.

注意到 $f(0)f(1)=(-\frac{8}{27})(\frac{1}{27})<0$, 所以介值定理保证在 $[0,1]$ 中有一个解. 根据例 1.2, 对分 20 步应足以保证 6 位精度.

事实上, 不用计算机也容易检验 $r=\frac{2}{3}=0.666\,666\,666\cdots$ 是一个根:

$$f(2/3)=\frac{8}{27}-2\left(\frac{4}{9}\right)+\left(\frac{4}{3}\right)\left(\frac{2}{3}\right)-\frac{8}{27}=0.$$

对分法能得到多少位数字? 出人意料地, 在计算 $f(0.666\,664\,1)=0$ 时, 对分法在 16 步后就停止了, 见表 1-7. 如果我们关心 6 位或更多位的精度, 这就是一次严重的失败. 图 1-7 说明了这个困难. 只要涉及 IEEE 双精度, 就有准确根 $r=\frac{2}{3}$的许多10^{-5}之内的浮点数被函数 $f(x)$ 计算成机器零, 因此它们都有权称为根! 尽管函数 f 单调递增, 但图 1-7b 显示甚至 f 的双精度值的符号也经常是错的, 这就把问题弄得更糟.

表 1-7

i	a_i	$f(a_i)$	c_i	$f(c_i)$	b_i	$f(b_i)$
0	0.000 000 0	−	0.500 000 0	−	1.000 000 0	+
1	0.500 000 0	−	0.750 000 0	+	1.000 000 0	+
2	0.500 000 0	−	0.625 000 0	−	0.750 000 0	+
3	0.625 000 0	−	0.687 500 0	+	0.750 000 0	+
4	0.625 000 0	−	0.656 250 0	+	0.687 500 0	+
5	0.656 250 0	−	0.671 875 0	+	0.687 500 0	+
6	0.656 250 0	−	0.664 062 5	−	0.671 875 0	+
7	0.664 062 5	−	0.667 968 8	+	0.671 875 0	+
8	0.664 062 5	−	0.666 015 6	−	0.667 968 8	+
9	0.666 015 6	−	0.666 992 2	+	0.667 968 8	+
10	0.666 015 6	−	0.666 503 9	−	0.666 992 2	+
11	0.666 503 9	−	0.666 748 0	+	0.666 992 2	+
12	0.666 503 9	−	0.666 626 0	−	0.666 748 0	+
13	0.666 626 0	−	0.666 687 0	+	0.666 748 0	+
14	0.666 626 0	−	0.666 656 5	−	0.666 687 0	+
15	0.666 656 5	−	0.666 671 8	+	0.666 687 0	+
16	0.666 656 5	−	0.666 664 1	0	0.666 671 8	+

图 1-7 表明问题不在于对分法, 而在于在根附近双精度算法不能足够精确地计算函数 f. 任何依靠这种计算机算法的其他解法都注定要失败. 对于这个例子, 甚至 16 位精度都不能检验候选解是否精确到 6 位. ◀

为了说明这并不是对分法造成的, 我们用 MATLAB 的最强大的多功能求根法, `fzero.m`. 本章后面将讨论其细节. 至于现在, 我们只需给它提供这个函数和初始估计. 不会有比这更幸运的事了.

```
>> fzero('x.^3-2*x.^2+4*x/3-8/27',1)
ans =
   0.66666250845989
```

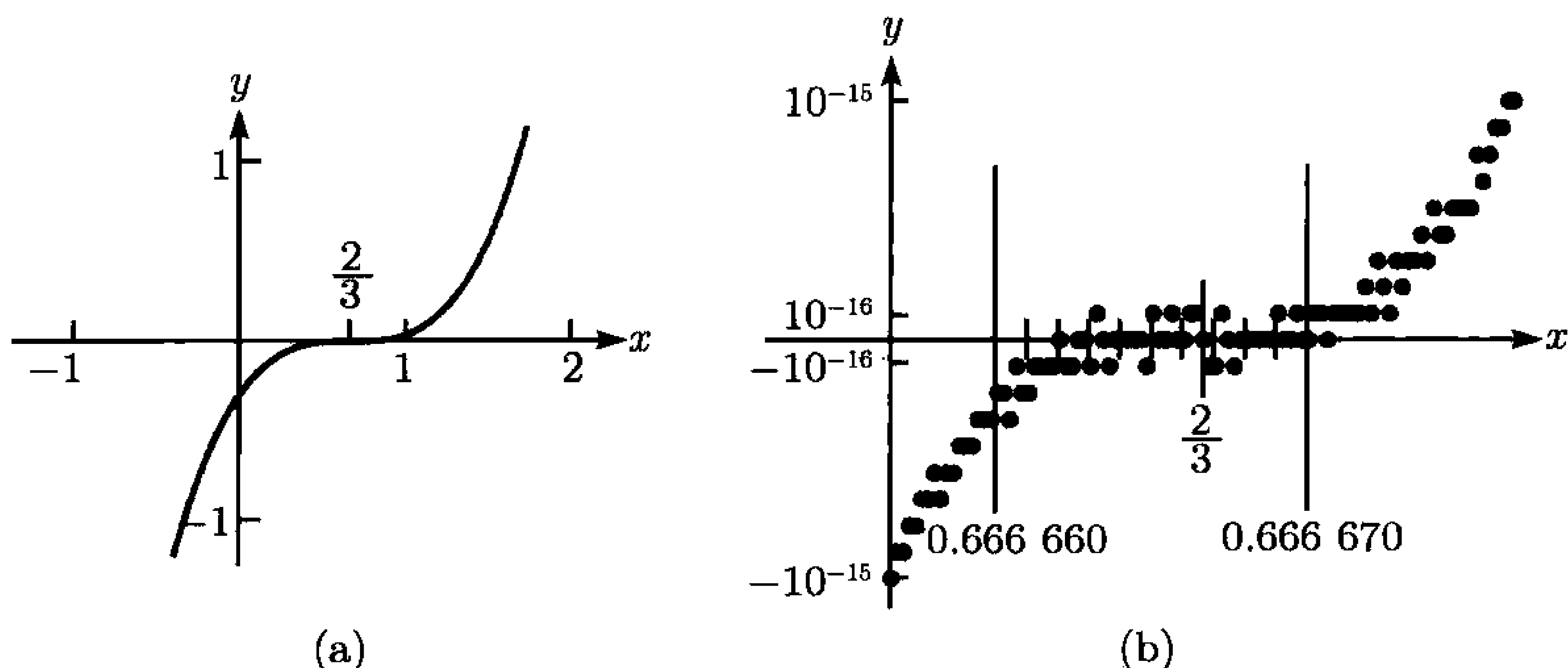


图 1-7 重根附近函数的形状: (a) $f(x)=x^3-2x^2+\frac{4}{3}x-\frac{8}{27}$ 的图; (b) 在根 $r=\frac{2}{3}$ 附近, (a) 的放大. 有许多浮点数, 它们离 $\frac{2}{3}$ 在 10^{-5} 之内, 它们都是计算机意义上的根. 我们由微积分可知只有 $\frac{2}{3}$ 是根

对于这个例子, 所有方法都不能得到多于 5 位精确数字, 其原因根据图 1-7 来看是清楚的. 在双精度计算中, 任何方法仅有的信息是函数值. 如果计算机运算表明函数在不是根的地方等于零, 这种方法就无法再次运用. 另一种说明困难的方法是指出, 若只涉及 y 轴, 那么近似解就能够尽可能地靠近解, 但在 x 轴方向并不特别靠近.

这些观察结果引出了几个关键的定义.

定义 1.8 假设 f 是一个函数, r 是一个根, 即它满足 $f(r)=0$. 假设 x_c 是 r 的近似值. 对于求根问题, 近似值 x_c 的**后向误差**(backward error) 是 $|f(x_c)|$, 而**前向误差**(forward error) 是 $|r-x_c|$.

"后向" 和 "前向" 的用法可能需要作一些解释. 我们的观点是把求解过程作为中心. 问题就是输入, 而解就是输出:

$$\text{定义问题的数据资料} \rightarrow \boxed{\text{求解过程}} \rightarrow \boxed{\text{解}}$$

本章的 "问题" 就是单变量的方程, "求解过程"(solution process) 就是解方程的一种算法:

$$\text{方程} \rightarrow \boxed{\text{方程解法}} \rightarrow \boxed{\text{解}}$$

后向误差是在左边或者输入 (问题数据) 一边. 我们需要用输出的近似值 x_c 来改变问题 (函数 f), 使方程平衡. 改变的量就是后向误差, 其值为 $|f(x_c)|$. 前向误差是在右边或者输出 (问题的解) 一边的误差. 我们需要改变近似解使它更准确. 改变的量就是前向误差, 其值为 $|r - x_c|$.

例 1.7 的困难是, 根据图 1-7, 后向误差接近 $\varepsilon_{\text{mach}} \approx 2.2 \times 10^{-6}$, 而前向误差近似于 10^{-5}. 双精度数在低于机器 ε 的相对误差时不能可靠地计算出来. 因为后向误差与可靠性都不能进一步减小, 前向误差也不能减小.

例 1.7 相当特殊, 因为函数在 $r = \frac{2}{3}$ 处有 3 重根. 注意

$$f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27} = \left(x - \frac{2}{3}\right)^3,$$

这是一个重根的例子.

定义 1.9 假设 r 是可微函数 f 的一个根, 即假设 $f(r) = 0$. 那么, 如果 $0 = f(r) = f'(r) = f''(r) = \cdots = f^{(m-1)}(r)$, 但是 $f^{(m)}(r) \neq 0$, 我们就说 f 在 r 处有 m **重根**. 如果重数大于 1, 我们就说 f 在 r 处有**重根**. 如果重数是 1, 则称这个根是**单根**.

例如, $f(x) = x^2$ 在 $r = 0$ 处有一个重数为 2 的根或者二重根, 这是因为 $f(0) = 0, f'(0) = 2(0) = 0$, 但是 $f''(0) = 2 \neq 0$. 同样, $f(x) = x^3$ 在 $r = 0$ 处有一个重数为 3 的根或者三重根, 而 $f(x) = x^m$ 在 $r = 0$ 有一个 m 重根. 例 1.7 在 $r = \frac{2}{3}$ 处有一个重数为 3 的根或者三重根.

因为函数的图形在靠近重根处相对平坦, 近似解附近后向误差和前向误差之间存在巨大的差异. 垂直方向测量得到的后向误差常常比水平方向测量得到的前向误差要小得多.

例 1.8 函数 $f(x) = \sin x - x$ 在 $r = 0$ 处有一个三重根, 求近似根 $x_c = 0.001$ 的后向误差和前向误差.

因为

$$f(0) = \sin 0 - 0 = 0, \qquad f'(0) = \cos\theta - 1 = 0,$$
$$f''(0) = -\sin\theta - 0 = 0, \qquad f'''(0) = \cos\theta = -1,$$

所以 0 处的根的重数是 3.

前向误差是 $FE = |r - x_c| = 10^{-3}$. 后向误差是常数, 它要被加到 $f(x)$ 使 x_c 成为一个根, 即

$$BE = |f(x_c)| = |\sin(0.001) - 0.001| \approx 1.666\,7 \times 10^{-10}. \qquad ◀$$

后向误差和前向误差与方程解法的停止准则有关. 目的是求根 r 使之满足 $f(r) = 0$. 假设算法产生了近似解 x_c, 我们怎样确定它是否足够令人满意?

我们想到有两种可能：(1) 使 $|x_c-r|$ 小, (2) 使 $|f(x_c)|$ 小. 在 $x_c=r$ 的情形下, 不用作决定：两种检查方法是相同的. 然而, 我们很少能幸运地遇到这种情形. 在更典型的情形中, 方法 (1) 和 (2) 是不同的, 而且它们对应于前向误差和后向误差.

前向误差合适还是后向误差合适依赖于问题的环境. 如果我们用对分法, 两种误差都很容易看出来. 对于近似根 x_c, 可以通过计算 $f(x_c)$ 的值求出后向误差, 而前向误差不会比目前区间长度的一半更大. 对于 FPI, 因为没有括起来的区间, 所以我们的选择更有限. 和前面一样, 后向误差被认为是 $f(x_c)$, 但是, 要知道前向误差就需要知道真正的根, 而它正是我们试图去求的.

方程解法的停止准则可以基于前向误差也可以基于后向误差. 还有可能相关的其他停止准则, 例如对于运算时间的限制. 停止准则的选取要根据问题的上下文而定.

函数在重根附近是平坦的, 这是因为导数 f' 在那里等于零. 因此, 在分离重根时像我们已说明的那样可以预计会遇到一些麻烦. 但重数仅是冰山一角. 如 1.3.2 节所示, 在看不到重根的地方可能产生类似的困难.

1.3.2 Wilkinson 多项式

只有单根但在数值上很难确定的著名例子由 J. Wilkinson[7] 给出. **Wilkinson多项式**形如

$$W(x)=(x-1)(x-2)\cdots(x-20), \tag{1.19}$$

乘出来就是

$$\begin{aligned} W(x)=&x^{20}-210x^{19}+20\,615x^{18}-1\,256\,850x^{17}+53\,327\,946x^{16}-1\,672\,280\,820x^{15}\\ &+40\,171\,771\,630x^{14}-756\,111\,184\,500x^{13}+11\,310\,276\,995\,381x^{12}\\ &-135\,585\,182\,899\,530x^{11}+1\,307\,535\,010\,540\,395x^{10}-10\,142\,299\,865\,511\,450x^{9}\\ &+63\,030\,812\,099\,294\,896x^{8}-311\,333\,643\,161\,390\,640x^{7}\\ &+1\,206\,647\,803\,780\,373\,360x^{6}-3\,599\,979\,517\,947\,607\,200x^{5}\\ &+8\,037\,811\,822\,645\,051\,776x^{4}-12\,870\,931\,245\,150\,988\,800x^{3}\\ &+13\,803\,759\,753\,640\,704\,000x^{2}-8\,752\,948\,036\,761\,600\,000x\\ &+2\,432\,902\,008\,176\,640\,000. \end{aligned} \tag{1.20}$$

它的根是 1~20 的整数. 然而, 当 $W(x)$ 按它非因子形式 (1.20) 定义时, 它的求值因几乎相等的大数相抵消而受到影响. 为了看清对于求根的影响, 通过表示成非因子形式 (1.20) 来定义 MATLAB 的 m 文件`wilkpoly.m`, 或者如果你的 MATLAB 版本中有符号运算功能, 便可以把它定义得更简单些.

```
function y=wilkpoly(s)
syms x
f=1;
for i=1:20
  f=f*(x-i);
end
f=collect(f);
y=subs(f,x,s);
```

此外, 我们将使用 MATLAB 的fzero命令. 为了使它尽可能简化, 我们给它提供一个实根 $x=16$ 作为初始估计.

```
>> fzero('wilkpoly', 16)
ans=
  16.01468030580458
```

MATLAB 的双精度运算甚至对单根 $r=16$ 也不能得到正确的两位小数. 这并非是算法的缺陷——不论是fzero还是对分法, 它们都与不动点迭代或者其他任何浮点方法一样有着相同的问题. 关于这个多项式, Wilkinson[8] 在 1984 年写道: "就我而言, 我认为它是我在作为数值分析人员的生涯中最受伤害的经验." $W(x)$ 的根显然是整数 $x=1,\cdots,20$. 使 Wilkinson 大为惊奇的是, 由于存储系数中很小的相对误差使根产生了巨大的误差放大, 这一点恰在上面的求解中已经看到.

如果使用因子形式 (1.19) 而不是 (1.20), 那么求 Wilkinson 多项式的精确根的困难就不会出现. 当然, 如果多项式在我们开始之前就因式分解了, 那么就不需要计算根了.

1.3.3 求根的灵敏度

Wilkinson 多项式和有三重根的例 1.7, 因为类似的原因而产生困难: 方程的小的浮点误差转化成根的大误差. 如果输入的小误差 (此时方程有解) 导致输出或解的大误差, 那么称这个问题是**灵敏的**(sensitive). 本节将量化灵敏度并介绍误差放大因子和条件数的概念.

为了了解是什么造成了这种误差放大, 我们将建立一个公式来预测当方程改变时, 根变化多大. 假设问题是求 $f(x)=0$ 的根 r, 但是对输入作一个小的改变 $\varepsilon g(x)$, 这里 ε 很小. 设 Δr 是相应的根的改变, 所以

$$f(r+\Delta r)+\varepsilon g(r+\Delta r)=0.$$

把 f 和 g 展成一次 Taylor 多项式, 即

$$f(r)+(\Delta r)f'(r)+\varepsilon g(r)+\varepsilon(\Delta r)g'(r)+O((\Delta r)^2)=0,$$

这里我们用了 "大 O" 记号 $O((\Delta r)^2)$, 它表示包括 $(\Delta r)^2$ 及 Δr 的更高次幂. 对于小的 $\Delta r, O((\Delta r)^2)$ 项可忽略不计, 从而得到

$$\Delta(r)(f'(r)+\varepsilon g'(r)) = -f(r)-\varepsilon g(r) = -\varepsilon g(r)$$

或者

$$\Delta r \approx \frac{-\varepsilon g(r)}{f'(r)+\varepsilon g'(r)} \approx -\varepsilon\frac{g(r)}{f'(r)},$$

假设与 $f'(r)$ 相比 ε 很小, 而且特别地 $f'(r)\neq 0$.

根的灵敏度公式

假设 r 是 $f(x)$ 的单根而且 $r+\Delta r$ 是 $f(x)+\varepsilon g(x)$ 的根. 如果 $\varepsilon \ll f'(r)$, 那么

$$\Delta r \approx -\frac{\varepsilon g(r)}{f'(r)}. \tag{1.21}$$

例 1.9 估计 $P(x)=(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)-10^{-6}x^7$ 的最大根.

令 $f(x)=(x-1)(x-2)(x-3)(x-4)(x-5)(x-6), \varepsilon=-10^{-6}$ 以及 $g(x)=x^7$. 如果没有 $\varepsilon g(x)$ 这一项, 那么最大根是 $r=6$. 问题是当加上这一项后, 根变化多大?

由灵敏度公式得到

$$\Delta r \approx -\frac{6^7\varepsilon}{5!} = -2\ 332.8\varepsilon.$$

意即在 $f(x)$ 中相对较小的输入误差 ε 对输出的根的影响被放大了 2000 多倍. 我们估计 $P(x)$ 的最大根是 $r+\Delta r = 6-2\ 332.8\varepsilon = 6.002\ 332\ 8$. 对 $P(x)$ 用`fzero`, 我们得到准确值 6.002 326 8. ◀

例 1.9 中的估计足以说明在求根问题中误差是如何传播的. 该问题数据中第 6 位数字上的误差造成了答案中第 3 位数字的误差, 即由于因子 2 332.8 而丢失了 3 位小数. 给这个因子取名很有用, 对产生近似解 x_c 的一般算法, 我们定义它的

$$\textbf{误差放大因子} = \frac{\text{相对前向误差}}{\text{相对后向误差}}.$$

前向误差表示使 x_c 准确的解的改变, 对于求根问题是 $|x_c-r|$. 后向误差表示使 x_c 为准确解的输入的改变. 依据我们所研究问题的灵敏度而有较广泛的选择. 在本节开始使用的选择是通过 $|f(x_c)|$ 改变常数项, 相应于灵敏度公式 (1.21) 中的 $g(x)=1$. 更一般地, 输入数据的任何改变可以用作后向误差, 譬如在例 1.9 中选择 $g(x)=x^7$. 求根的误差放大因子是

$$\text{误差放大因子} = \left|\frac{\frac{\Delta r}{r}}{\varepsilon g(r)/g(r)}\right| \approx \left|\frac{-\varepsilon g(r)/(rf'(r))}{\varepsilon}\right| = \frac{|g(r)|}{|rf'(r)|}, \tag{1.22}$$

它在例 1.9 中就是 $6^7/(5!6)=388.8$.

例 1.10 用根的灵敏度公式研究 Wilkinson 多项式中 x^{15} 项的改变对根 $r = 16$ 的影响. 求出这个问题的误差放大因子.

定义扰动函数 $W_\varepsilon(x) = W(x) + \varepsilon g(x)$, 这里 $g(x) = -1.672\ 280\ 820x^{15}$. 注意 $W'(16) = 15!4!$(见习题 7). 用公式 (1.21), 根的改变近似为

$$\Delta r \approx \frac{16^{15} \times 1\ 672\ 280\ 820\varepsilon}{15!4!} \approx 6.143\ 2 \times 10^{13}\varepsilon. \tag{1.23}$$

实际地讲, 我们从第 0 章知道, 对每一个存储的数必须假定一个机器 ε 阶的相对误差. 机器 $\varepsilon_{\text{mach}}$ 的 x^{15} 项的相对改变将造成根 $r = 16$ 以

$$\Delta r \approx (6.143\ 2 \times 10^{13})(\pm 2.22 \times 10^{-16}) \approx \pm 0.013\ 6$$

变化到 $r + \Delta r \approx 16.013\ 6$, 接近 1.3.2 节代码显示的结果. 当然, Wilkinson 多项式中许多其他 x 的幂次项各自作出了自己的贡献, 所以完全地描述对根改变的影响是很复杂的. 然而, 灵敏度公式让我们看清了误差的巨大放大过程.

最后根据 (1.22), 我们把误差放大因子按下式计算

$$\frac{|g(r)|}{|rf'(r)|} = \frac{16^{15} \times 1\ 672\ 280\ 820}{15! \times 4! \times 16} \approx 3.8 \times 10^{12}. \qquad ◀$$

误差放大因子的重要性是, 它告诉我们 16 位数字的运算精度在输入到输出的过程中损失了多少位. 对于一个误差放大因子为 10^{12} 的问题, 我们预测损失了 16 位数字中的 12 位, 在根中仅留下 4 位有效数字, 这就是 Wilkinson 近似 $x_c = 16.014\cdots$ 的情形.

上述误差放大的例子说明了求根对于特殊的输入改变的灵敏度. 问题的灵敏度可能大小不一, 这取决于如何指定输入的改变. 问题的**条件数**(condition number) 定义为所有的输入改变或者至少是所有指定类型的改变情况下最大的误差放大. 条件数大的问题称为**病态的**(ill-conditioned), 而条件数接近 1 的问题称为**良态的**(well-conditioned). 在第 2 章讨论矩阵时, 我们将回到这个概念.

亮点　条件作用

这是第一次出现条件数的概念, 它是误差放大的度量. 数值分析研究的是算法, 它把定义问题的数据取为输入, 而把得到的答案作为输出. 条件数指的是理论问题本身固有的这种放大的部分, 而不考虑用于求它的特别算法.

注意到误差放大因子仅仅度量由问题产生的放大是十分重要的. 和条件作用一起, 有一个平行的概念, 即稳定性. 它指的是由算法对小的输

入误差的放大,而不是问题本身. 如果一种算法总能提供带有小的后向误差的近似解,则称其为稳定的. 如果问题是良态的,并且算法是稳定的,我们就能够期望后向误差和前向误差都是小的.

习题 1.3

1. 求以下函数的前向误差和后向误差,这里的根是 $\frac{3}{4}$,而近似根是 $x_c = 0.74$:
   (a) $f(x) = 4x - 3$; (b) $f(x) = (4x-3)^2$; (c) $f(x) = (4x-3)^3$; (d) $f(x) = (4x-3)^{\frac{1}{3}}$.
2. 求以下函数的前向误差和后向误差,这里的根是 $\frac{1}{3}$,而近似根是 $x_c = 0.333\ 3$:
   (a) $f(x) = 3x - 1$; (b) $f(x) = (3x-1)^2$; (c) $f(x) = (3x-1)^3$; (d) $f(x) = (3x-1)^{\frac{1}{3}}$.
3. (a) 求 $f(x) = 1 - \cos x$ 的根 $r = 0$ 的重数;
   (b) 求近似根 $x_c = 0.000\ 1$ 的前向误差和后向误差.
4. (a) 求 $f(x) = x^2 \sin x^2$ 的根 $r = 0$ 的重数;
   (b) 求近似根 $x_c = 0.01$ 的前向误差和后向误差.
5. 找出求线性函数 $f(x) = ax - b$ 的根的前向误差和后向误差之间的关系.
6. 设 n 是正整数. 定义正数 A 的第 n 个根的方程是 $x^n - A = 0$. (a)求根的重数; (b) 证明: 对有小的前向误差的 n 次近似根,常数项的后向误差近似地是前向误差的 $nA^{(n-1)/n}$ 倍.
7. 设 $W(x)$ 是 Wilkinson 多项式. (a) 证明 $W'(16) = 15!4!$; (b) 对 $W'(j)$ 求出类似的公式,这里 j 是 1 和 20 之间的一个整数.
8. 令 $f(x) = x^n - ax^{n-1}$,并设 $g(x) = x^n$. (a) 对小的 ε,用灵敏度公式给出 $f_\varepsilon(x) = x^n - ax^{n-1} + \varepsilon x^n$ 的非零根的预测; (b) 求出非零根并与这个预测比较.

计算机问题 1.3

1. 设 $f(x) = \sin x - x$. (a) 求根 $r = 0$ 的重数; (b) 取初始估计 $x = 0.1$,用 MATLAB 的`fzero`命令确定根的位置. `fzero`给出的前向误差和后向误差是什么?
2. 对 $f(x) = \sin x^3 - x^3$ 执行计算机问题 1.
3. (a) 用`fzero`求 $f(x) = 2x\cos x - 2x + \sin x^3$ 在 $[-0.1, 0.2]$ 上的根. 报告前向误差和后向误差.
   (b) 取初始区间 $[-0.1, 0.2]$,运行对分法,求出尽可能多的正确数字,并报告你的结果.
4. (a) 对常数 ε,用 (1.21) 近似求解 $f_\varepsilon(x) = (1+\varepsilon)x^3 - 3x^2 + x - 3$ 在 3 附近的根.
   (b) 取 $\varepsilon = 10^{-3}$,求出实际根并与 (a) 比较.
5. 用 (1.21) 去近似 $f(x) = (x-1)(x-2)(x-3)(x-4) - 10^{-6}x^6$ 在 $r = 4$ 附近的根. 求误差放大因子. 用`fzero`检验你的近似.
6. 用 MATLAB 命令`fzero`求 Wilkinson 多项式在靠近 $x = 15$ 处的根, x^{15} 项的系数中有 $\varepsilon = 2 \times 10^{-15}$ 的相对改变,使得系数的绝对值稍微变大一点. 与 (1.21) 得出的预测进行比较.

1.4 Newton 法

Newton 法也叫做 Newton-Raphson 法, 其收敛速度通常比我们前面已看到的线性收敛方法要快得多. 图 1-8 中给出了 Newton 法的几何图形. 要求出 $f(x)=0$ 的根, 初始估计 x_0 是给定的, 画出函数 f 在 x_0 处的切线. 切线朝着根的方向贴近函数向下到 x 轴. 切线与 x 轴的交点就是近似根, 但是如果 f 弯曲, 很可能不准确. 因此这一步要反复进行.

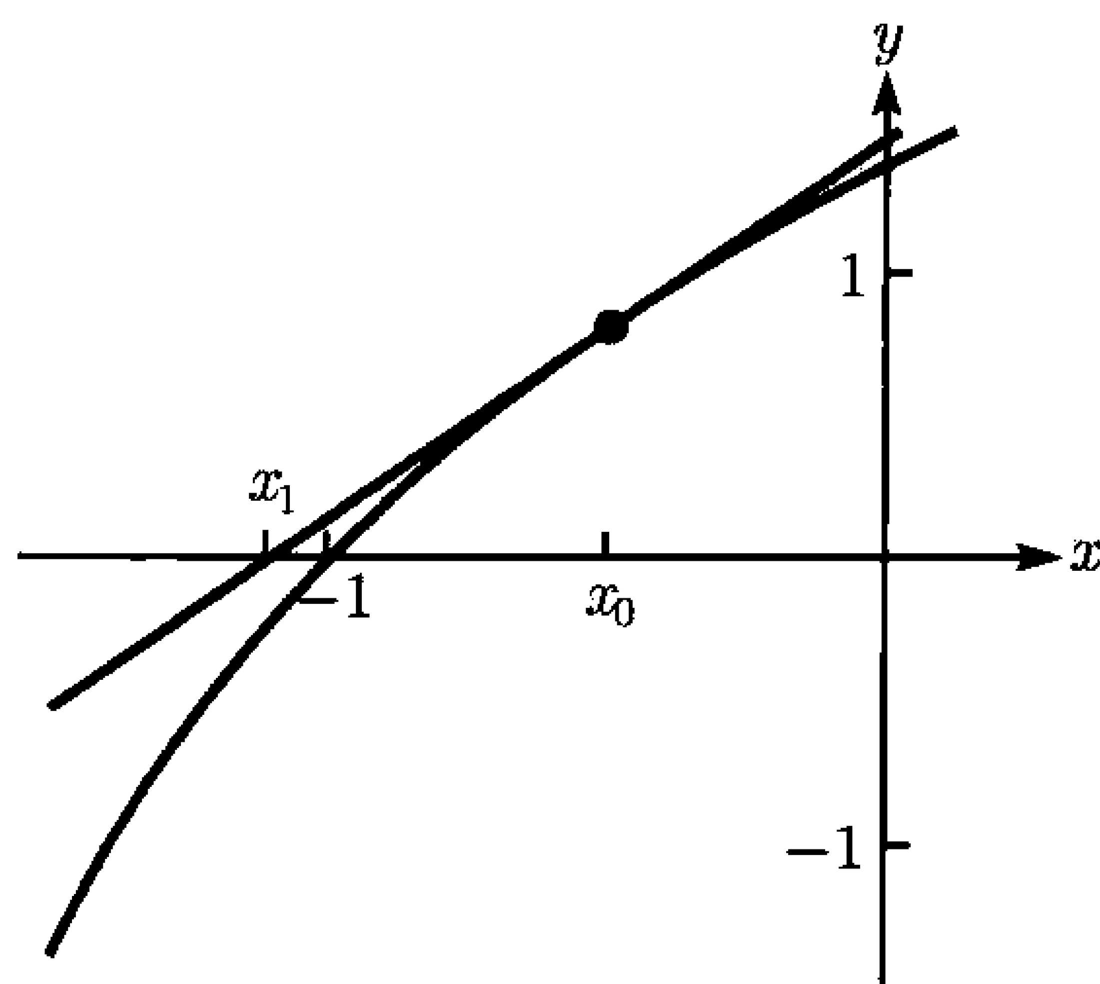


图 1-8 Newton 法的一步. 从 x_0 出发, 画出曲线 $y=f(x)$ 的切线. 它与 x 轴的交点是 x_1, 是根的下一个近似

从这个几何图形可以建立 Newton 方法的代数公式. 切线在 x_0 的斜率由导数 $f'(x_0)$ 给出, 切线上的一点是 $(x_0, f(x_0))$. 直线方程的点斜式是 $y-f(x_0)=f'(x_0)(x-x_0)$, 所以求切线与 x 轴的交点等同于在直线方程中取 $y=0$:

$$
\begin{aligned}
f'(x_0)(x-x_0) &= 0-f(x_0), \\
x-x_0 &= -\frac{f(x_0)}{f'(x_0)}, \\
x &= x_0-\frac{f(x_0)}{f'(x_0)},
\end{aligned}
$$

解 x 给出了根的近似值, 我们把它叫做 x_1. 下一步是重复整个过程, 从 x_1 开始产生 x_2, 如此等等, 得到如下迭代公式:

Newton 法

$$
\begin{aligned}
x_0 &= \text{初始估计}, \\
x_{i+1} &= x_i-\frac{f(x_i)}{f'(x_i)}, \qquad i=0,1,2,\cdots.
\end{aligned}
$$

例 1.11 对方程 $x^3+x-1=0$, 求 Newton 法的公式.

因为 $f'(x)=3x^2+1$, 公式由下面给出

$$x_{i+1} = x_i - \frac{x_i^3 + x_i - 1}{3x_i^2 + 1} = \frac{2x_i^3 + 1}{3x_i^2 + 1}.$$

从初始点 $x_0 = -0.7$ 开始迭代这个公式得到

$$x_1 = \frac{2x_0^3 + 1}{3x_0^2 + 1} = \frac{2 \times (-0.7)^3 + 1}{3 \times (-0.7)^2 + 1} \approx 0.127\,1,$$

$$x_2 = \frac{2x_1^3 + 1}{3x_1^2 + 1} \approx 0.957\,7.$$

图 1-9 从几何上表示了这几步. 表 1-8 给出了以后各步.

表 1-8

i	x_i	$e_i = \|x_i - r\|$	e_i/e_{i-1}^2
0	−0.700 000 00	1.382 327 80	
1	0.127 125 51	0.555 203 00	0.290 6
2	0.957 678 12	0.275 350 32	0.893 3
3	0.734 827 79	0.052 499 99	0.692 4
4	0.684 591 77	0.002 263 97	0.821 4
5	0.682 332 17	0.000 004 37	0.852 7
6	0.682 327 80	0.000 000 00	0.854 1
7	0.682 327 80	0.000 000 00	

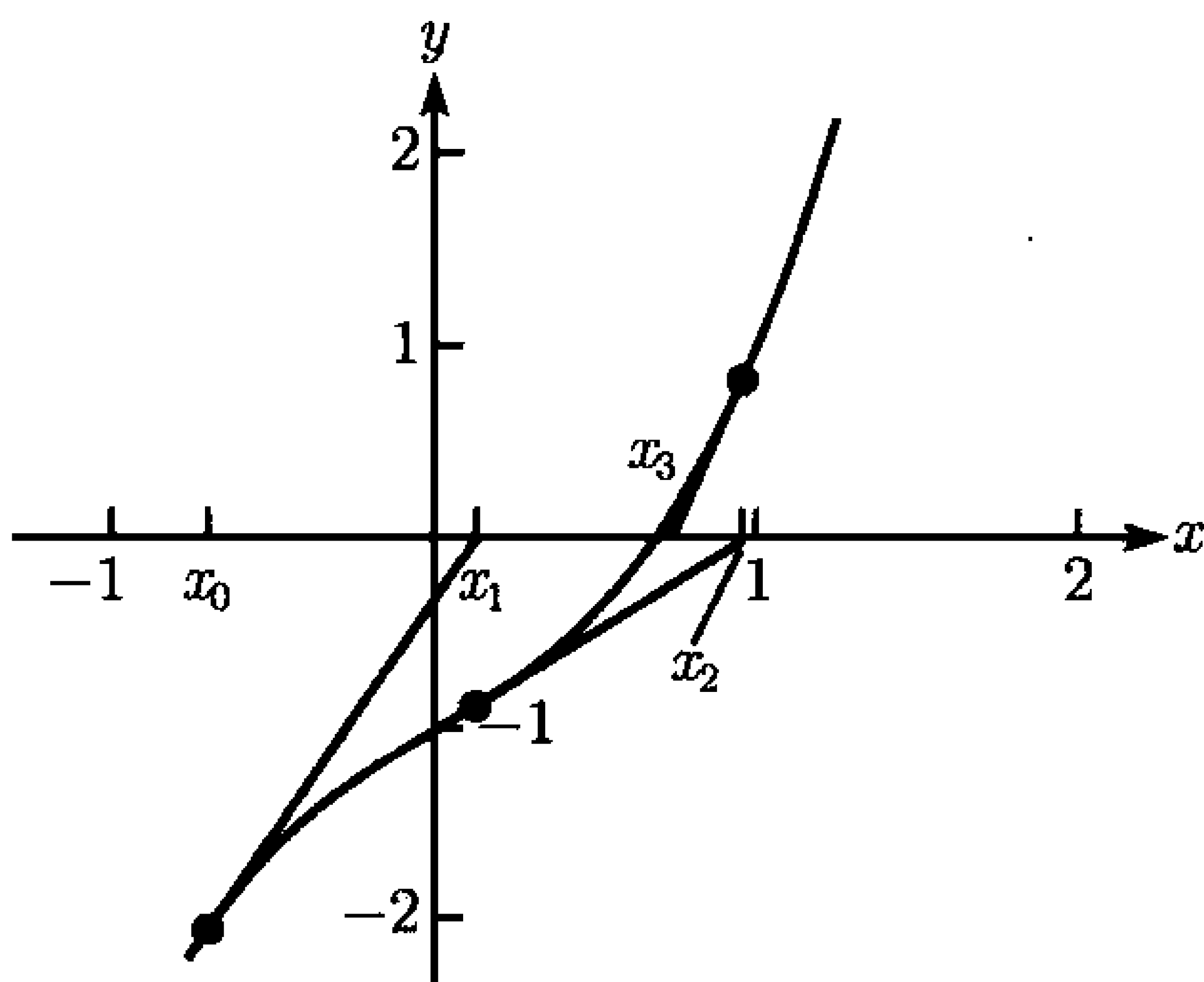


图 1-9 Newton 法的三步. 从 $x_0 = -0.7$ 开始的例 1.11 的图解, Newton 法迭代解沿着切线画出, 该方法可以认为收敛于根

仅仅 6 步后, 根达到 6 位精确数字. 关于误差以及它怎样快地变小, 我们还可以多说一点. 注意在表中, 收敛性一旦开始成立, 每一次迭代在 x_i 中正确的位数近似地加倍. 这是 "二次收敛" 方法的特点, 我们将在下一节看到. ◀

1.4.1 Newton 法的二次收敛性

例 1.11 中的收敛速度大大快于对分法和不动点迭代中的线性收敛速度. 我们需要新的定义.

定义 1.10 设 e_i 表示一种迭代方法在第 i 步后的误差. 如果

$$M = \lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} < \infty,$$

那么这种迭代是二次收敛的.

定理 1.11 设 f 是二次连续可微函数, 并且 $f(r)=0$. 如果 $f'(r)\neq 0$, 那么 Newton 法局部而且二次收敛于 r. 第 i 步的误差满足

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = M,$$

其中

$$M = \left|\frac{f''(r)}{2f'(r)}\right|.$$

证 要证明局部收敛性, 只要注意 Newton 法是不动点迭代的特殊形式, 这里

$$g(x) = x - \frac{f(x)}{f'(x)},$$

其导数为

$$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$

因为 $g'(r)=0$, 根据定理 1.6, Newton 法是局部收敛的.

为证明二次收敛性, 下面用另一种方法来导出 Newton 法. 这一次我们密切关注每一步的误差. 误差在这里表示正确的根和目前的最佳估计之间的差.

Taylor 公式给出了函数在给定点和附近另一个点的值之间的差. 对于这两个点, 我们将选用根 r 和 i 步之后的当前估计 x_i, 并在展开两项之后停止, 取一个余项:

$$f(r) = f(x_i) + (r-x_i)f'(x_i) + \frac{(r-x_i)^2}{2}f''(c_i).$$

这里 c_i 在 x_i 和 r 之间. 因为 r 是根, 我们有

$$\begin{aligned}
0 &= f(x_i) + (r-x_i)f'(x_i) + \frac{(r-x_i)^2}{2}f''(c_i) \\
-\frac{f(x_i)}{f'(x_i)} &= r - x_i + \frac{(r-x_i)^2}{2}\frac{f''(c_i)}{f'(x_i)},
\end{aligned}$$

这里假设 $f'(x_i)\neq 0$. 重新整理后, 我们就能把下一步的 Newton 迭代解与根进行比较:

$$\begin{aligned}
x_i - \frac{f(x_i)}{f'(x_i)} - r &= \frac{(r-x_i)^2}{2}\frac{f''(c_i)}{f'(x_i)}, \\
x_{i+1} - r &= e_i^2 \frac{f''(c_i)}{2f'(x_i)}, \\
e_{i+1} &= e_i^2 \left|\frac{f''(c_i)}{2f'(x_i)}\right|.
\end{aligned} \tag{1.24}$$

在这个等式中, 我们已定义第 i 步的误差是 $e_i = x_i - r$. 因为 c_i 落在 r 和 x_i 之间, 它会如 x_i 那样收敛于 r , 并且

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = \left|\frac{f''(r)}{2f'(r)}\right|.$$

这就是二次收敛性的定义.

可以把已建立的误差公式 (1.24) 看成

$$e_{i+1} \approx M e_i^2, \tag{1.25}$$

假定 $f'(r) \neq 0$, 这里的 $M = \left|\frac{f''(r)}{2f'(r)}\right|$. 由于估计值 x_i 趋近于 r, 又因为 c_i 在 x_i 和 r 之间, 所以当 Newton 法收敛时, 这个近似值会更好. 这个误差公式倒是可以与线性收敛方法的公式 $e_{i+1} \approx S e_i$ 进行比较, 对于 FPI, 这里的 $S = |g'(r)|$, 而对于对分法, $S = \frac{1}{2}$.

虽然 S 的值对线性收敛方法是关键的, 但是 M 值的重要度却要低一些. 因为这个公式包含了前一个误差的平方. 一旦误差显著地小于 1, 平方将使它进一步减小, 并且只要 M 不太大, 根据公式 (1.25) 误差也会减小.

回到例 1.11, 我们可以分析输出表来说明误差的比率, 右边的一列表示比 $\frac{e_i}{e_{i-1}^2}$, 根据 Newton 法误差公式 (1.25), 当根收敛时, 它应趋近于 M. 对于 $f(x) = x^3 + x - 1$, 其导数是 $f'(x) = 3x^2 + 1$ 及 $f''(x) = 6x$, 在 $x_c \approx 0.682\,3$ 处求值得到 $M \approx 0.85$, 这与表的右列中的误差比是一致的.

有了对 Newton 法的新理解, 我们就能够全面地解释例 1.6 的平方根算法. 设 a 是正数, 考虑用 Newton 法求 $f(x) = x^2 - a$ 的根. 迭代是

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^2 - a}{2x_i} = \frac{x_i^2 + a}{2x_i} = \frac{x_i + \frac{a}{x_i}}{2}. \tag{1.26}$$

对任意的 a, 这就是例 1.6 中的方法.

为了研究其收敛性, 在根 $\sqrt{a}$ 处求导数值:

$$\begin{aligned} f'(\sqrt{a}) &= 2\sqrt{a}, \\ f''(\sqrt{a}) &= 2. \end{aligned} \tag{1.27}$$

因为 $f'(\sqrt{a}) = 2\sqrt{a} \neq 0$, 所以 Newton 法二次收敛, 而且收敛速度是

$$e_{i+1} \approx M e_i^2, \tag{1.28}$$

这里 $M = 2/(2 \cdot 2\sqrt{a}) = 1/(2\sqrt{a})$.

1.4.2 Newton 法的线性收敛性

定理 1.11 没有说 Newton 法总是二次收敛. 为了二次收敛有意义我们需要用 $f'(r)$ 相除. 这个假设被证实是关键的. 下面的例 1.12 说明 Newton 法并不是二次收敛的情形.

例 1.12 用 Newton 法求 $f(x) = x^2$ 的根.

这可以看成一个小问题, 因为我们知道有一个根 $r = 0$. 但是把一种新方法用到我们完全了解的例子中去常常是有启发性的. Newton 法的公式是

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^2}{2x_i} = \frac{x_i}{2}$$

出人意料的结果是: Newton 法简化成除以 2. 因为根是 $r = 0$, 所以对初始估计 $x_0 = 1$, 有 Newton 迭代的表 1-9.

表 1-9

i	x_i	$e_i = \lvert x_i - r\rvert$	e_i/e_{i-1}
0	1.000	0.000	
1	0.500	0.500	0.500
2	0.250	0.250	0.500
3	0.125	0.125	0.500
⋮	⋮	⋮	⋮

Newton 法确实收敛于根 $r = 0$. 误差公式是 $e_{i+1} = e_i/2$. 所以收敛是线性的, 其收敛比例常数 $S = \frac{1}{2}$. ◀

如下例所示, 对任意的正整数 m, x^m 有类似结果成立.

例 1.13 用 Newton 法求 $f(x) = x^m$ 的根.

Newton 公式是

$$x_{i+1} = x_i - \frac{x_i^m}{mx_i^{m-1}} = \frac{m-1}{m}x_i,$$

同样, 仅有的根是 $r = 0$, 所以定义 $e_i = |x_i - r| = x_i$, 得到

$$e_{i+1} = Se_i$$

这里 $S = \frac{m-1}{m}$. ◀

这个例子表明了 Newton 法在重根处的一般性质. 注意到重根的定义 1.9 等价于 $f(r) = f'(r) = 0$, 恰好在这种情形下我们不能对 Newton 法误差公式求导. 对重根有一种独立的误差公式. 我们看到的单项式重根是一般情况的代表, 如定理 1.12 中所总结的那样.

定理 1.12 假设在 $[a,b]$ 上 $(m+1)$ 次连续可微函数 f 在 r 处有 m 重根, 那么 Newton 法局部收敛到 r, 而且第 i 步的误差 e_i 满足

$$\lim_{i\to\infty}\frac{e_{i+1}}{e_i}=S, \tag{1.29}$$

这里 $S=\frac{m-1}{m}$.

亮点 收敛性

式 (1.28) 和式 (1.29) 表示 Newton 法中可能的两种不同的收敛到根的速度. 在单根情形下, $f'(r)\neq 0$, 收敛是二次的, 或者是满足式 (1.28) 的快速收敛. 在重根情形下 $f'(r)=0$, 收敛是线性的, 并且满足式 (1.29). 在后一种线性收敛情形下, 较慢的收敛速度使 Newton 法与对分法和 FPI 属同一类别.

例 1.14 求 $f(x)=\sin x+x^2\cos x-x^2-x$ 的根 $r=0$ 的重数, 并估计 Newton 法收敛到 6 位正确小数所需要的步数 (设 $x_0=1$).

容易验证

$$\begin{aligned}
f(x)&=\sin x+x^2\cos x-x^2-x,\\
f'(x)&=\cos x+2x\cos x-x^2\sin x-2x-1,\\
f''(x)&=-\sin x+2\cos x-4x\sin x-x^2\cos x-2
\end{aligned}$$

它们在 $r=0$ 处都等于 0. 但是三阶导数

$$f'''(x)=-\cos x-6\sin x-6x\cos x+x^2\sin x, \tag{1.30}$$

满足 $f'''(0)=-1$, 所以根 $r=0$ 是三重根, 意即重数 $m=3$. 根据定理 1.12, Newton 法以 $e_{i+1}\approx 2e_i/3$ 而线性收敛.

设初始估计 $x_0=1$, 我们有 $e_0=1$. 接近收敛时, 每一步的误差递减 1/3. 因此要得到误差在 6 位小数之内或者小于 0.5×10^{-6} 的大概步数, 可以通过解

$$\begin{aligned}
\left(\frac{2}{3}\right)^n &< 0.5\times10^{-6}\\
n &> \frac{\log_{10}(0.5)-6}{\log_{10}(2/3)}\approx 35.78
\end{aligned} \tag{1.31}$$

而得到. 大约将需要 36 步. 最初 20 步见表 1-10.

注意右列中的误差比收敛到预测的 $\frac{2}{3}$. ◀

如果事先知道根的重数, 那么可以通过小的修改而提高 Newton 法的收敛性.

表 1-10

i	x_i	$e_i = \|x_i - r\|$	e_i/e_{i-1}
1	1.000 000 000 000 00	1.000 000 000 000 00	
2	0.721 590 239 860 75	0.721 590 239 860 75	0.721 590 239 860 75
3	0.521 370 951 820 40	0.521 370 951 820 40	0.722 530 493 096 77
4	0.375 308 308 590 76	0.375 308 308 590 76	0.719 848 904 662 50
5	0.268 363 490 527 13	0.268 363 490 527 13	0.715 048 093 485 61
6	0.190 261 613 699 24	0.190 261 613 699 24	0.708 969 813 015 61
7	0.133 612 505 326 19	0.133 612 505 326 19	0.702 256 764 926 86
8	0.092 925 286 725 17	0.092 925 286 725 17	0.695 483 454 174 55
9	0.064 039 266 777 34	0.064 039 266 777 34	0.689 147 906 174 74
10	0.043 778 062 160 09	0.043 778 062 160 09	0.683 612 795 135 59
11	0.029 728 055 524 23	0.029 728 055 524 23	0.679 062 846 946 49
12	0.020 081 683 737 77	0.020 081 683 737 77	0.675 512 857 590 09
13	0.013 512 127 304 17	0.013 512 127 304 17	0.672 858 286 217 86
14	0.009 065 795 643 30	0.009 065 795 643 30	0.670 937 702 052 49
15	0.006 070 292 922 63	0.006 070 292 922 63	0.669 581 927 662 31
16	0.004 058 851 096 27	0.004 058 851 096 27	0.668 641 719 271 13
17	0.002 711 303 677 93	0.002 711 303 677 93	0.667 997 818 500 81
18	0.001 809 959 662 50	0.001 809 959 662 50	0.667 560 656 240 29
19	0.001 207 723 844 67	0.001 207 723 844 67	0.667 265 613 533 25
20	0.000 805 633 071 49	0.000 805 633 071 49	0.667 067 289 464 60

定理 1.13 如果函数 f 在区间 $[a, b]$ 上 $(m+1)$ 次连续可微, 且含有重数 $m > 1$ 的根 r, 那么修正Newton法

$$x_{i+1} = x_i - \frac{mf(x_i)}{f'(x_i)} \tag{1.32}$$

局部且二次收敛于 r.

回到例 1.14, 我们可以用修正 Newton 法来实现二次收敛. 5 步后对 $r = 0$ 的收敛已取到大约 8 位数字的精度, 见表 1-11.

表 1-11

i	x_i
0	1.000 000 000 000 00
1	0.164 770 719 582 24
2	0.016 207 337 711 44
3	0.000 246 541 437 74
4	0.000 000 060 722 72
5	−0.000 000 006 332 50

表中有几点要注意. 首先, 对近似根的二次收敛性是可以看得出来的, 因为在每一步近似根的正确位数大体上加倍, 直到第 4 步. 第 6 步及以后各步都和第 5 步

完全一样. Newton 法缺少对机器精度的收敛性, 我们在 1.3 节中已熟悉其原因. 检验得 0 是重根. 虽然后向误差被 Newton 法压近 $\varepsilon_{\max}$, 但前向误差等于 x_i, 并大了几个量级.

和 FPI 一样, Newton 法可能不收敛于根. 下面的例题正好说明其可能不收敛的行为之一.

例 1.15 取初始估计 $x_0 = 1$, 对 $f(x) = -x^4 + 3x^2 + 2$ 应用 Newton 法.

因为 $f(x)$ 连续, 在 $x = 0$ 处是正的, 并且对大的正数和大的负数 x, 它趋于负无穷大, 所以这个函数有根. 然而如图 1-10 所示, 对初始估计将求不出根. Newton 公式是

$$x_{i+1} = x_i - \frac{-x_i^4 + 3x_i^2 + 2}{-4x_i^3 + 6x_i}. \tag{1.33}$$

代入得到 $x_1 = -1$, 然后又得到 $x_2 = 1$. 在这个例题中, Newton 法交替地在两个不是根的 1 和 -1 之间变动, 因此求根失败.

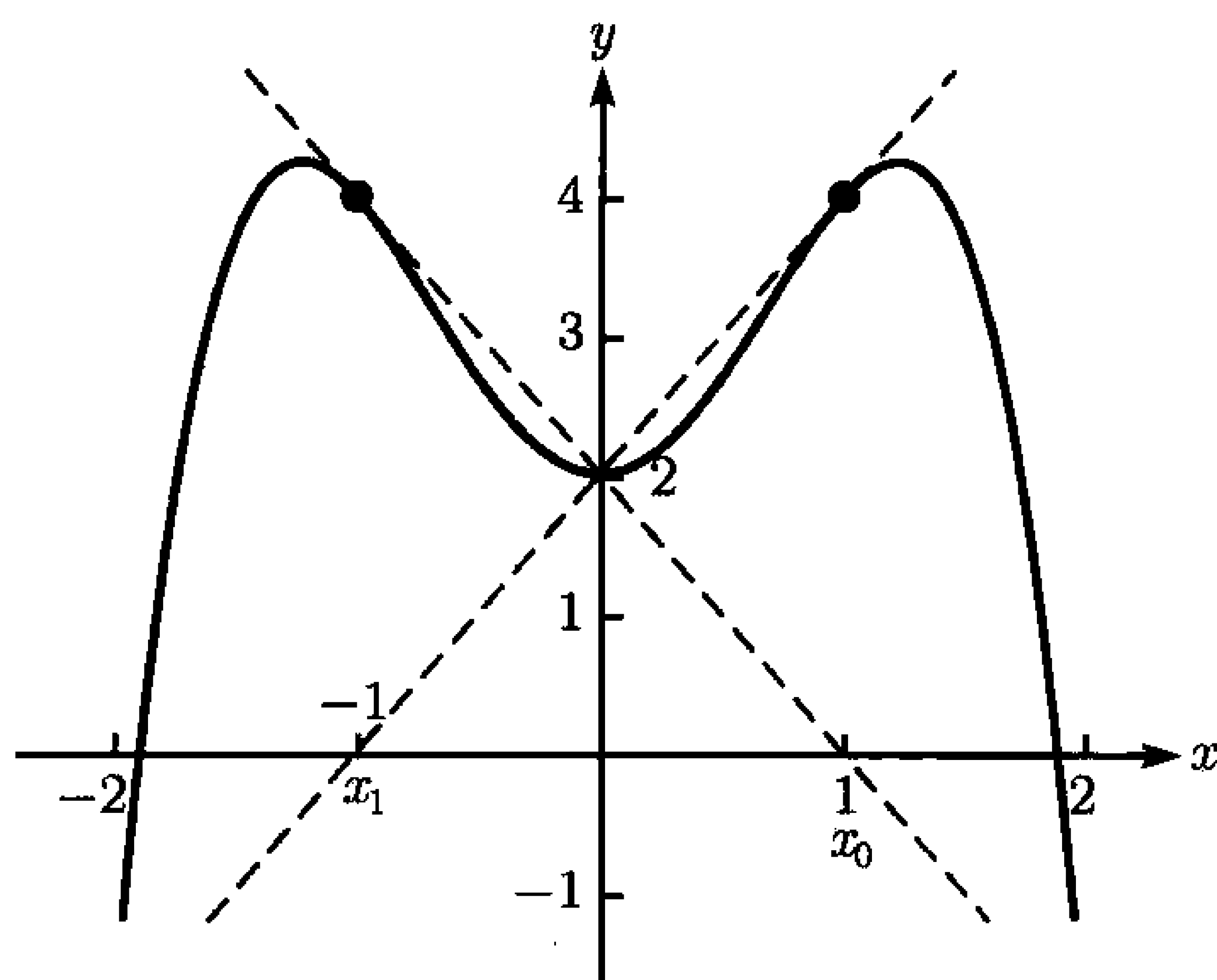


图 1-10 在例 1.15 中 Newton 法失败. 迭代交替于 1 和 -1 之间, 因此不收敛于根

Newton 法可能在第一种情形失败. 显然, 如果在任一迭代步 $f'(x_i) = 0$, 则该方法就不能继续. 还有迭代发散到无穷大 (见习题 6) 或模拟随机数生成算法 (见计算机问题 13) 的其他例子. 尽管并不是每一个初始估计都收敛于根, 但是定理 1.11 和定理 1.12 保证了对于每一个根周围的初始估计, 在其邻域内收敛于根.

习题 1.4

1. 取初始估计 $x_0 = 0$, 应用 Newton 法进行两步:

   (a) $x^3 + x - 2 = 0$; (b) $x^4 - x^2 + x - 1 = 0$; (c) $x^2 - x - 1 = 0$.

2. 取初始估计 $x_0 = 1$, 应用 Newton 法进行两步:

   (a) $x^3 + x^2 - 1 = 0$; (b) $x^2 + \frac{1}{x+1} - 3x = 0$; (c) $5x - 10 = 0$.

3. 当 Newton 法收敛到给定的根时, 用定理 1.11 或定理 1.12 通过前面的误差 e_i 来估计误差 e_{i+1}. 收敛是线性的还是二次的?

(a) $x^5 - 2x^4 + 2x^2 - x = 0$; $r = -1, r = 0, r = 1$;

(b) $2x^4 - 5x^3 + 3x^2 + x - 1 = 0$, $r = -\frac{1}{2}, r = 1$.

4. 如在习题 3 中那样估计 e_{i+1}:

(a) $32x^3 - 32x^2 - 6x + 9 = 0, r = -\frac{1}{2}, r = \frac{3}{4}$; (b) $x^3 - x^2 - 5x - 3 = 0; r = -1, r = 3$.

5. 考虑方程 $8x^4 - 12x^3 + 6x^2 - x = 0$. 对两个解 $x = 0$ 及 $x = \frac{1}{2}$ 中的每一个, 不进行计算确定是对分法还是 Newton 法将收敛较快 (譬如说精确到 8 位).

6. 设计一个函数 f 以及初始估计, 使得对它们的 Newton 法发散.

7. 设 $f(x) = x^4 - 7x^3 + 18x^2 - 20x + 8$.Newton 法是否二次收敛于根 $r = 2$? 求 $\lim\limits_{i\to\infty} \frac{e_{i+1}}{e_i}$, 这里 e_i 表示第 i 步的误差.

8. 证明: 用于 $f(x) = ax + b$ 的 Newton 法一步就收敛.

9. 证明: 对 $f(x) = x^2 - A$ 用 Newton 法便产生例 1.6 的迭代.

10. 对 $f(x) = x^3 - A$ 用 Newton 法, 求所产生的不动点迭代, 见习题 1.2.10.

11. 用 Newton 法来产生一个二次收敛方法, 计算正数 A 的第 n 个根这里 n 是正整数. 证明二次收敛性.

12. 如果初始估计是 $x_0 = 1$, 对 $f(x) = \frac{1}{x}$ 用 Newton 法求 x_{50}.

13. (a) 函数 $f(x)=x^3-4x$ 在 $r=2$ 处有根. 如果应用 Newton 法 4 步之后的误差 $e_i=x_i-r$ 是 $e_4=10^{-6}$, 估计 e_5.

(b) 对根 $r=0$ 做与 (a) 相同的问题. (提醒: 平常所用的公式未必有用.)

计算机问题 1.4

1. 每个方程有一个根. 用 Newton 法把根近似到 8 位有效小数.

(a) $x^3 = 2x + 2$; (b) $e^x + x = 7$; (c) $e^x + \sin x = 4$.

2. 每个方程有一个根. 用 Newton 法把根近似到 8 位有效小数.

(a) $x^5 + x = 1$; (b) $\sin x = 6x + 5$; (c) $\ln x + x^2 = 3$.

3. 用 Newton 法尽可能精确地求出仅有的根, 并求出根的重数. 然后用修正 Newton 法来二次收敛于这个根. 报告每种方法得到的最佳近似的前向误差和后向误差.

(a) $f(x) = 27x^3 + 54x^2 + 36x + 8$; (b) $f(x) = 36x^4 - 12x^3 + 37x^2 - 12x + 1$.

4. 对下列方程, 执行计算机问题 3 中的步骤:

(a) $f(x) = 2e^{x-1} - x^2 - 1$; (b) $f(x) = \ln(3 - x) + x - 2$.

5. 一个具有半球形圆顶、10 米高的直圆柱形粮仓的体积是 400 立方米, 求粮仓的底半径, 精确到 4 位有效小数.

6. 顶点为一个半球形凹洞、10 厘米高的圆锥容纳了 60 立方厘米的冰淇淋, 求凹洞的半径, 精确到 4 位有效小数.

7. 考虑在区间 $[-2, 2]$ 上的函数 $f(x) = e^{\sin^3 x} + x^6 - 2x^4 - x^3 - 1$. 画出在这个区间上函数的图形, 并求全部 3 个根, 精确到 6 位有效小数. 确定哪些根二次收敛, 并求线性收敛的根的重数.

8. 对下面的函数 $f(x)$, 执行计算机问题 7 的步骤:

$f(x) = 94\cos^3 x - 24\cos x + 1\ 77\sin^2 x - 108\sin^4 x - 72\cos^3 x\sin^2 x - 65, x \in [0, 3]$.

9. 用 Newton 法求函数 $f(x) = 14xe^{x-2} - 12e^{x-2} - 7x^3 + 20x^2 - 26x + 12$在区间 $[0, 3]$ 上

的两个根. 对每个根打印迭代序列、误差 e_i 及收敛到非零极限的相应的误差比 $\frac{e_{i+1}}{e_i^2}$ 或者 $\frac{e_{i+1}}{e_i}$. 把这个极限与定理 1.11 得到的预期值 M 或者定理 1.12 得到的 S 进行比较.

10. 设 $f(x) = 54x^6 + 45x^5 - 102x^4 - 69x^3 + 35x^2 + 16x - 4$. 画出在区间 $[-2, 2]$ 上的函数图形, 并用 Newton 法求这个区间内的全部 5 个根. 确定对哪些根 Newton 法线性收敛, 而对哪些根收敛是二次的.

11. 对于低温和低压的气体, 理想气体定律是 $PV = nRT$, 这里 P 是压强 (大气压), V 是体积 (升), T 是温度 (开尔文), n 是气体的摩尔数以及 $R = 0.082\ 057\ 8$ 是摩尔气体常数. Van der Waals 方程

$$\left(P + \frac{n^2 a}{V^2}\right)(V - nb) = nRT$$

覆盖了非理想情况, 在那里, 这些假设都不成立. 用理想气体定律计算初始估计, 接着通过对 Van der Waals 方程用 Newton 方法求在 320K 和 15 atm(大气压) 的压强下摩尔氧气的体积. 对于氧气, $a = 1.36\text{L}^2\text{-atm/mole}^2$(升 ^{2}- 大气压/摩尔 ^{2}), $b = 0.003\ 183$ L/mole(升/摩尔), 用 3 位有效数字表示你的初始估计和解.

12. 利用计算机问题 11 得到的资料, 求在 700K 及 20 个大气压下, 1 摩尔苯蒸气的体积. 对于苯, $a = 18.0\text{L}^2\text{-atm/mole}^2$, $b = 0.115\ 4$ L/mole.

13. **(a)** 求函数 $f(x) = (1 - 3/(4x))^{\frac{1}{3}}$ 的根; **(b)** 应用 Newton 法并画出前 50 次迭代. 产生混乱的轨迹是 Newton 法可能失败的另一种方式.

1.5 不用导数求根

除了重根之外, Newton 法比对分法和 FPI 方法收敛得快. 它能达到这种较快的速度是因为它用了更多的信息——特别是关于函数切线的信息, 而这一点来自函数的导数. 但在某些情形下, 导数可能得不到.

在这种情形下, 割线法是 Newton 法的好的替代. 它用一种称为割线的近似来代替切线, 而且几乎收敛得一样快. 割线法的变形用一条近似抛物线代替这条直线. 抛物线的轴要么是垂直的 (Muller 法) 要么是水平的 (反二次插值), 这一节最后将叙述 Brent 方法, 这种方法结合了迭代法和加括号法的最好优点.

1.5.1 割线法及其变形

割线法类似于 Newton 法, 但是用差商代替导数. 在几何上, 是用一条通过两个上次已知的猜测的直线来代替切线. "割线" 的交点是新的估计.

在当前的估计 x_i 处, 导数的近似是差商

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

在 Newton 法中, 直接把这种近似代替 $f'(x_i)$ 就得到割线法.

割线法

$$x_0, x_1 = \text{初始估计},$$
$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}, \qquad i = 1, 2, 3, \cdots,$$

与不动点迭代和 Newton 法不同, 割线法需要使用两个初始估计.

可以证明: 在割线法收敛到 r 并且 $f'(r) \neq 0$ 的假设下, 近似误差关系

$$e_{i+1} \approx \left| \frac{f''(r)}{2f'(r)} \right| e_i e_{i-1}$$

成立, 因此这就意味着

$$e_{i+1} \approx \left| \frac{f''(r)}{2f'(r)} \right|^{\alpha - 1} e_i^{\alpha}$$

这里 $\alpha = (1+\sqrt{2})/2 \approx 1.62$(见习题 6). 割线法对单根的收敛性称为**超线性的**(superlinear), 就是说它位于线性收敛和二次收敛方法之间.

例 1.16 取初始估计 $x_0 = 0, x_1 = 1$, 用割线法求 $f(x) = x^3 + x - 1$ 的根.

公式给出

$$x_{i+1} = x_i - \frac{(x_i^3 + x_i - 1)(x_i - x_{i-1})}{x_i^3 + x_i - (x_{i-1}^3 + x_{i-1})}. \tag{1.34}$$

从 $x_0 = 0$ 和 $x_1 = 1$ 开始, 我们计算

$$x_2 = 1 - \frac{1 \times (1 - 0)}{1 + 1 - 0} = \frac{1}{2}, \qquad x_3 = \frac{1}{2} - \frac{-\frac{3}{8} \times (\frac{1}{2} - 1)}{-\frac{3}{8} - 1} = \frac{7}{11},$$

如图 1-11 所示. 进一步迭代形成表 1-12

表 1-12

i	x_i	i	x_i
0	0.000 000 000 000 00	5	0.682 020 419 648 19
1	1.000 000 000 000 00	6	0.682 325 781 409 89
2	0.500 000 000 000 00	7	0.682 327 804 359 03
3	0.636 363 636 363 64	8	0.682 327 803 828 02
4	0.690 052 356 020 94	9	0.682 327 803 828 02

◀

有 3 种割线法的推广也是重要的. **试位法**(method of false position 或 regula falsi) 类似于对分法, 但是这里的中点被类割线法近似所代替. 给定含有根的区间 $[a, b]$(假定 $f(a)f(b) < 0$), 像在割线法中一样, 定义下一个点

$$c = \frac{bf(a) - af(b)}{f(a) - f(b)},$$

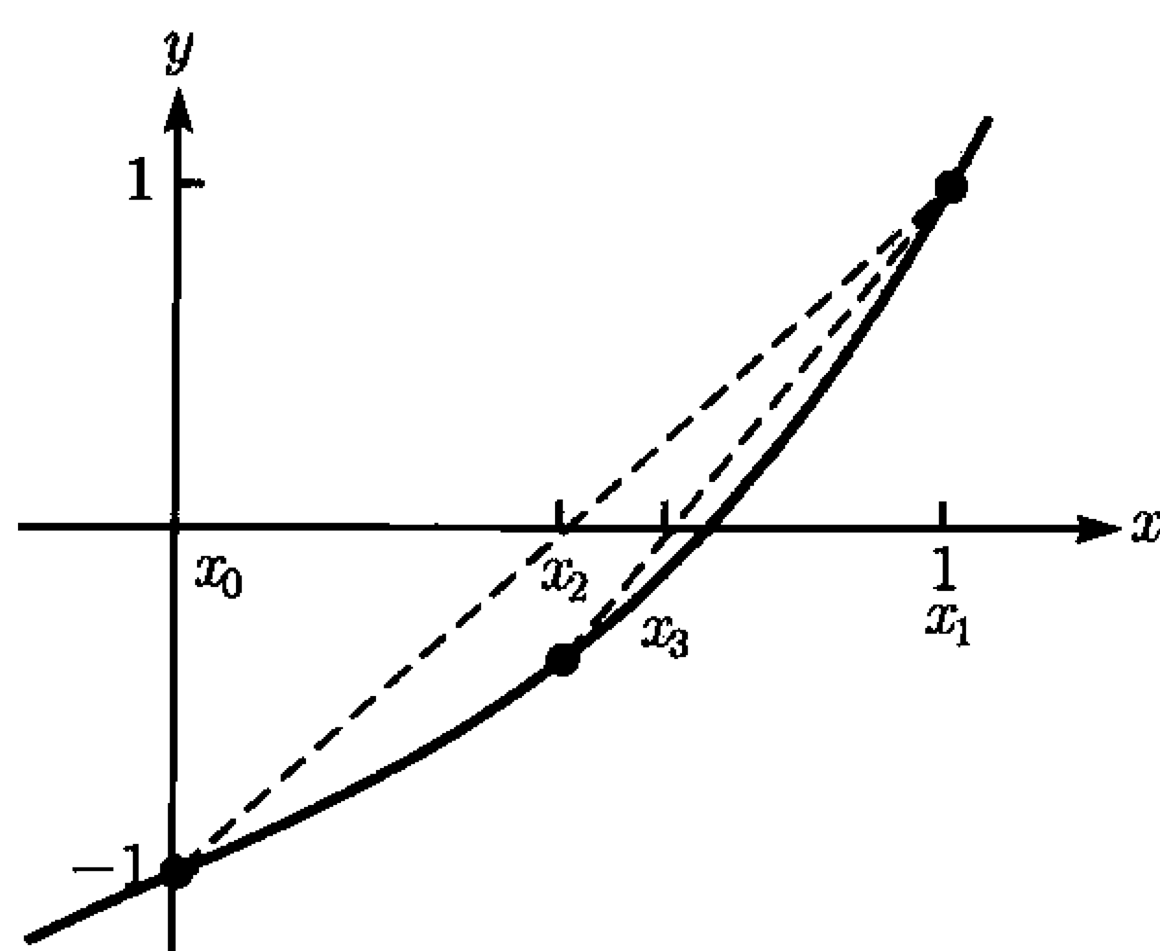


图 1-11 割线法的两步. 例 1.16 的图解. 从 $x_0=0$ 及 $x_1=1$ 开始, 割线法迭代解沿着割线画出

但是又不像割线法, 因为点 $(a, f(a))$ 和 $(b, f(b))$ 在 x 轴的两侧, 所以 c 点一定在 $[a, b]$ 中. 新区间或者选取 $[a, c]$ 或者选取 $[c, b]$, 根据 $f(a)f(c) < 0$, 还是 $f(c)f(b) < 0$, 因此新区间仍含有根.

试位法

给定区间 $[a, b]$, 使 $f(a)f(b) < 0$
for $i = 1, 2, 3, \cdots$
  $c = \frac{bf(a)-af(b)}{f(a)-f(b)}$
  if $f(c) = 0$ **stop end**
  if $f(a)f(c) < 0$
    $b = c$
  else
    $a = c$
  end
end

乍一看, 试位法似乎是对对分法和割线法作了改进, 吸收了每一种方法的最好性质. 然而, 对分法保证每一步都将不确定度减少 $\frac{1}{2}$, 试位法却不作这种承诺, 对某些例子可能收敛得非常慢.

例 1.17 用试位法在初始区间 $[-1, 1]$ 上求 $f(x) = x^3 - 2x^2 + \frac{3}{2}x$ 的根 $r = 0$.

给定 $x_0 = -1, x_1 = 1$ 作为初始区间, 我们计算新的点

$$x_2 = \frac{x_1 f(x_0) - x_0 f(x_1)}{f(x_0) - f(x_1)} = \frac{1 \times (-\frac{9}{2}) - (-1) \times \frac{1}{2}}{-\frac{9}{2} - \frac{1}{2}} = \frac{4}{5}$$

因为 $f(-1)f(\frac{4}{5}) < 0$, 所以新的区间是 $[x_0, x_2] = [-1, 0.8]$, 这就完成了第一步. 注意这个解的不确定度的递减因子远小于 $\frac{1}{2}$. 如图 1-12b 所示, 以后各步很缓慢地朝根 $x = 0$ 逼近. ◀

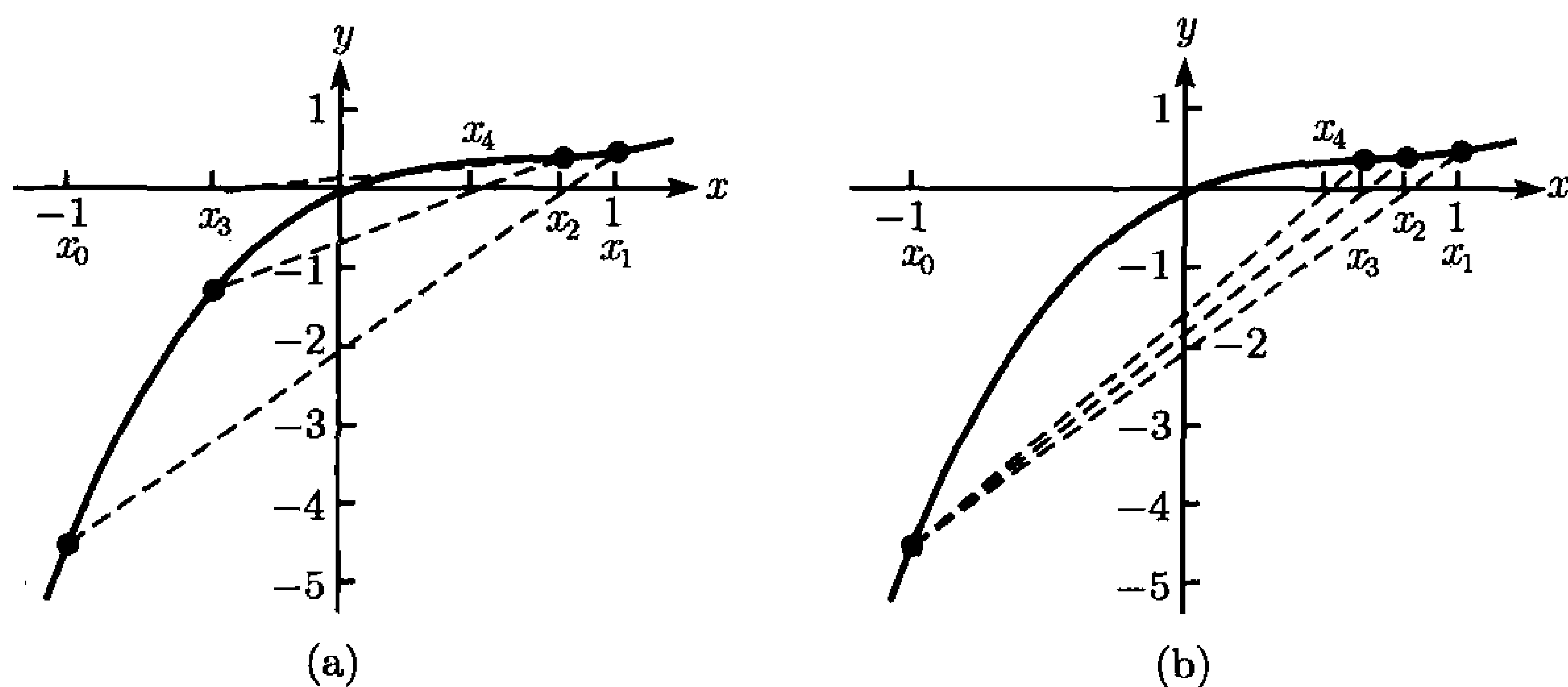


图 1-12 例 1.17 中的慢收敛. (a) 割线法; (b) 试位法, 都慢收敛于根 $r = 0$

Muller 方法是割线法在不同方向的推广. 不是通过前面两个点与 x 轴相交, 我们用前面 3 个点 x_0, x_1, x_2, 画出通过它们的抛物线 $y = p(x)$, 而且这条抛物线与 x 轴相交. 抛物线一般有 0 或者 2 个交点. 如果有两个交点, 则距离上一个点 x_2 最近的一个点被选作 x_3. 用二次公式确定这两种可能性是一件简单的事情. 如果抛物线与 x 轴不相交, 就有复数解. 这就得处理复数算法的软件来确定复根. 我们将不再深入考虑这个想法, 尽管关于这个方面的文献中有好几种来源.

反二次插值(Inverse Quadratic Interpolation, IQI) 是割线法对抛物线的推广. 虽然抛物线具有 $x = p(y)$ 的形式, 而不像在 Muller 方法中 $y = p(x)$ 的形式. 一个问题立刻得到解决: 这条抛物线将与 x 轴交于一个点, 所以从前 3 个估计 x_i, x_{i+1}, x_{i+2} 来求 x_{i+3} 还是很明确的.

通过 3 点 $(a, A), (b, B), (c, C)$ 的二次多项式 $x = P(y)$ 是

$$P(y) = a\frac{(y-B)(y-C)}{(A-B)(A-C)} + b\frac{(y-A)(y-C)}{(B-A)(B-C)} + c\frac{(y-A)(y-B)}{(C-A)(C-B)}. \tag{1.35}$$

这是 Lagrange 插值的一个例子, 它是第 3 章所研究的问题之一. 现在只要注意到 $P(A) = a, P(B) = b$ 及 $P(C) = c$ 就够了. 代入 $y = 0$ 就给出了抛物线与 x 轴交点的公式. 经过整理和代入, 我们有

$$P(0) = c - \frac{r(r-q)(c-b) + (1-r)s(c-a)}{(q-1)(r-1)(s-1)}, \tag{1.36}$$

这里 $q = f(a)/f(b), r = f(c)/f(b), s = f(c)/f(a)$.

对于 IQI, 取 $a = x_i, b = x_{i+1}, c = x_{i+2}$ 及 $A = f(x_i), B = f(x_{i+1}), C = f(x_{i+2})$, 则下一个估计 $x_{i+3} = P(0)$ 就是

$$x_{i+3} = x_{i+2} - \frac{r(r-q)(x_{i+2} - x_{i+1}) + (1-r)s(x_{i+2} - x_i)}{(q-1)(r-1)(s-1)}, \tag{1.37}$$

这里 $q = f(x_i)/f(x_{i+1}), r = f(x_{i+2})/f(x_{i+1})$, $s = f(x_{i+2})/f(x_i)$. 给定 3 个初始估计, IQI 方法通过迭代 (1.37) 就可以进行处理. 用新的估计 x_{i+3} 代替最老的估计 x_i.IQI 的另一种实现方式是用新的估计代替前面 3 个估计中后向误差最大的那一个.

图 1-13 把 Muller 方法与反二次插值法的几何图形作了比较. 由于是高阶插值, 所以这两种方法都比割线法收敛快. 第 3 章中有关于插值的更多内容. 割线法及其推广以及对分法的理念是 Brent 方法的关键要素. 这是 1.5.2 节的主题.

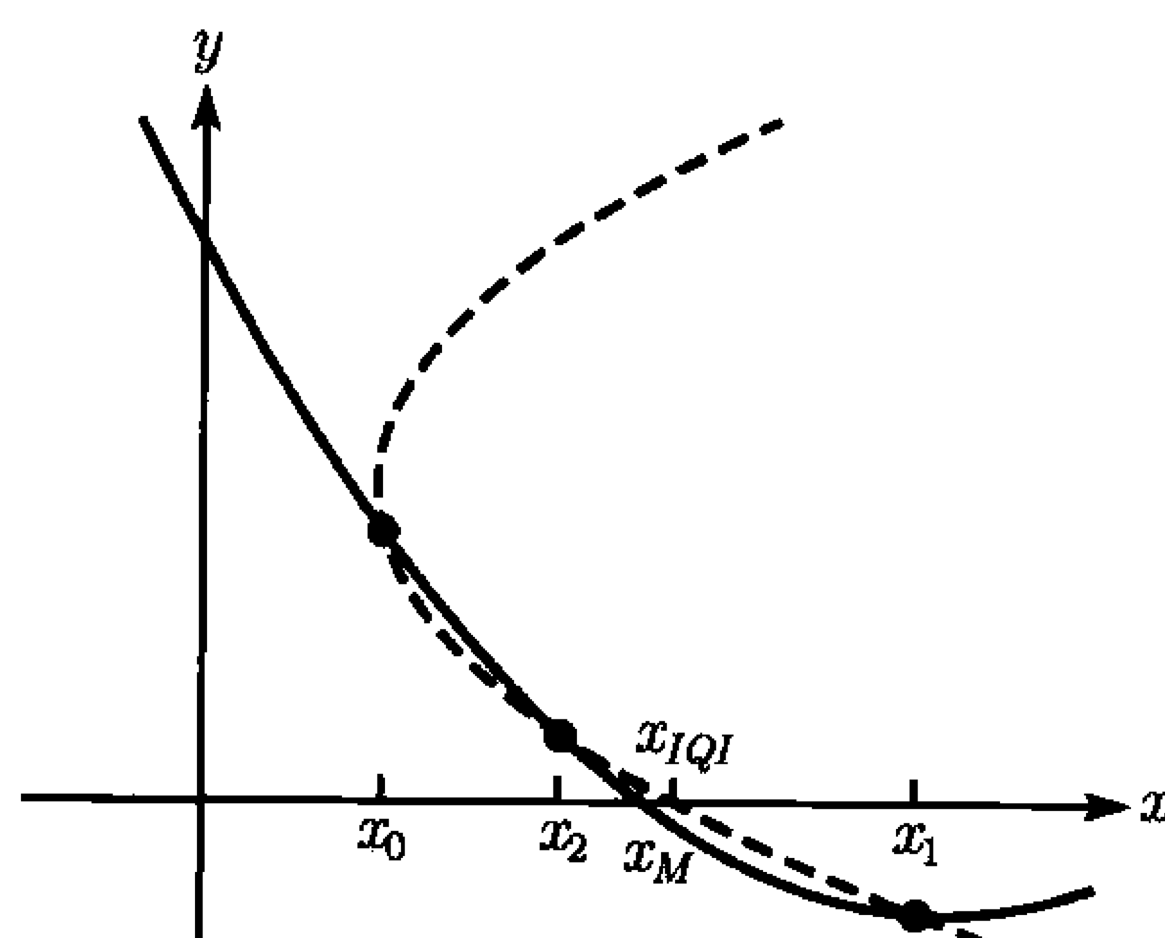


图 1-13 Muller 方法步与反二次插值步的比较. 前者由插值抛物线 $y = p(x)$ 确定, 后者由 $x = p(y)$ 确定

1.5.2 Brent 方法

Brent 方法是混合方法, 它利用以前介绍的部分解题技巧建立一种保留各自最有用性质的新方法. 它最希望把对分法保证收敛的性质与较复杂方法的快速收敛性质结合起来. 它最初是由 Dekker 和 Van Wijngaarden 在 20 世纪 60 年代提出的.

这种方法用于连续函数 f, 而且区间以 a 和 b 为界, 这里 $f(a)f(b) < 0$. Brent 方法记录在后向误差意义下最好的当前点 x_i, 以及包括根的区间 $[a_i, b_i]$. 粗略地讲, 尝试用反二次插值法, 如果 (1) 后向误差改进了, (2) 包含根的区间至少缩小一半, 那么就用结果代替 x_i, a_i, b_i 中的一个; 否则, 就尝试用割线法. 如果还是失败, 就按对分法的步骤, 保证其不确定度至少减半.

MATLAB 命令`fzero`实现了一种 Brent 方法, 包含了一个预处理步, 如果用户没有提供包含根的初始区间, `fzero`就会找一个比较好的初始区间. 停止准则属于混合的前向/后向误差类型. 当 x_i 到新的点 x_{i+1} 之间的改变小于 $2\varepsilon_{\max}\max(1, x_i)$ 或者当后向误差 $|f(x_i)|$ 达到机器零时, 算法就终止.

如果用户提供了初始区间, 那么就不启用预处理步. 使用下面这个命令会进入函数 $f(x) = x^3 + x - 1$ 和初始区间, 并且要求 MATLAB 显示每一次迭代的部分结果.

```
>> fzero('x^3+x-1',[0 1],optimset('Display','iter'))

 Func-count        x              f(x)            Procedure
     1              0               -1            initial
     2              1                1            initial
     3            0.5           -0.375            bisection
     4       0.636364        -0.105935            interpolation
     5       0.684910       0.00620153            interpolation
     6       0.682225     -0.000246683            interpolation
     7       0.682328     -5.43508e-007           interpolation
     8       0.682328     1.50102e-013            interpolation
     9       0.682328                0            interpolation
Zero found in the interval: [0, 1].

ans=

   0.68232780382802
```

另外, 命令

```
>> fzero('x^3+x-1', 1)
```

首先找出包含根的区间, 然后应用 Brent 方法, 从而求 $f(x)$ 在 $x=1$ 附近的根.

习题 1.5

1. 取初始估计 $x_0=1$ 及 $x_1=2$, 对方程执行割线法的两步.
   (a) $x^3=2x+2$; (b) $e^x+x=7$; (c) $e^x+\sin x=4$.
2. 取初始区间 $[1,2]$, 对习题 1 中的方程执行试位法的两步.
3. 对习题 1 中的方程执行反二次插值法的两步. 使用初始估计 $x_0=1, x_1=2$ 及 $x_2=0$, 并通过保持 3 个最新的迭代值进行更新.
4. 一位渔民想在温度为 40°F 的水底深处布置鱼网. 通过投入一条附有温度计的细绳, 她发现在水深 12 米处温度是 38°, 而在水深 5 米处温度是 46°. 由割线法确定温度是 40° 的水深的最佳估计.
5. 通过把 $y=0$ 代入式 (1.35), 推导出式 (1.36).
6. 如果割线法收敛到 $r, f'(r)\neq 0$, 而且 $f''(r)\neq 0$, 那么能够证明近似误差关系 $e_{i+1}\approx |f''(r)/(2f'(r))|e_ie_{i-1}$. 证明如下结论: 如果另加上条件, 对某个 $\alpha>0$, $\lim\limits_{i\to\infty} e_{i+1}/e_i^\alpha$ 存在, 并且不等于 0, 那么 $\alpha=(1+\sqrt{5})/2$ 并且 $e_{i+1}\approx|f''(r)/2f'(r)|^{\alpha-1}e_i^\alpha$.

计算机问题 1.5

1. 用割线法求习题 1 中每个方程的 (单个) 解.
2. 用试位法求习题 1 中每个方程的解.
3. 用反二次插值法求习题 1 中每个方程的解.
4. 设 $f(x)=54x^6+45x^5-102x^4-69x^3+35x^2+16x-4$. 画出在区间 $[-2,2]$ 上的函数图形, 并且用割线法求在区间内的所有 5 个根. 对哪一个根是线性收敛? 对哪一个根是超线性收敛?

5. 在习题 6 中你曾被问及, 对于在区间 $[-2,1]$ 上的 $f(x)=\frac{1}{x}$, 对分法的结果会是什么? 现在对这个问题应用`fzero`, 并比较结果.

6. (a) 如果要求`fzero`去求 $f(x)=x^2$ 靠近 1 处的根会发生什么 (不使用包含根的区间)? 解释这个结果. (b) 对 $f(x)=1+\cos x$ 求靠近 -1 处的根回答同样的问题.

实例检验 1 STEWART 平台的运动学

Stewart 平台包括 6 根可变长度的支柱或者棱柱铰 (prismatic joint), 支持一个有效负载 (pay load). 通常利用气功或液力传动来改变支柱的长度, 从而操纵棱柱铰. 像一个有 6 个自由度的机器人一样, Stewart 平台可以放在它能到达的三维空间中的任何一点和任一斜面处.

为了简单起见, 本实例考虑二维的 Stewart 平台. 它将模拟一个操纵装置, 其中包括一个三角形平台, 平台位于一个由 3 个支柱控制的固定平面中, 如图 1-14 所示. 里面的三角形表示平面型 Stewart 平台, 它的尺寸由 3 个长度 L_1, L_2, L_3 确定. 设 γ 表示与边 L_1 相对的角. 平台的位置由 3 个支柱的可变长度的 3 个数 p_1, p_2, p_3 所控制.

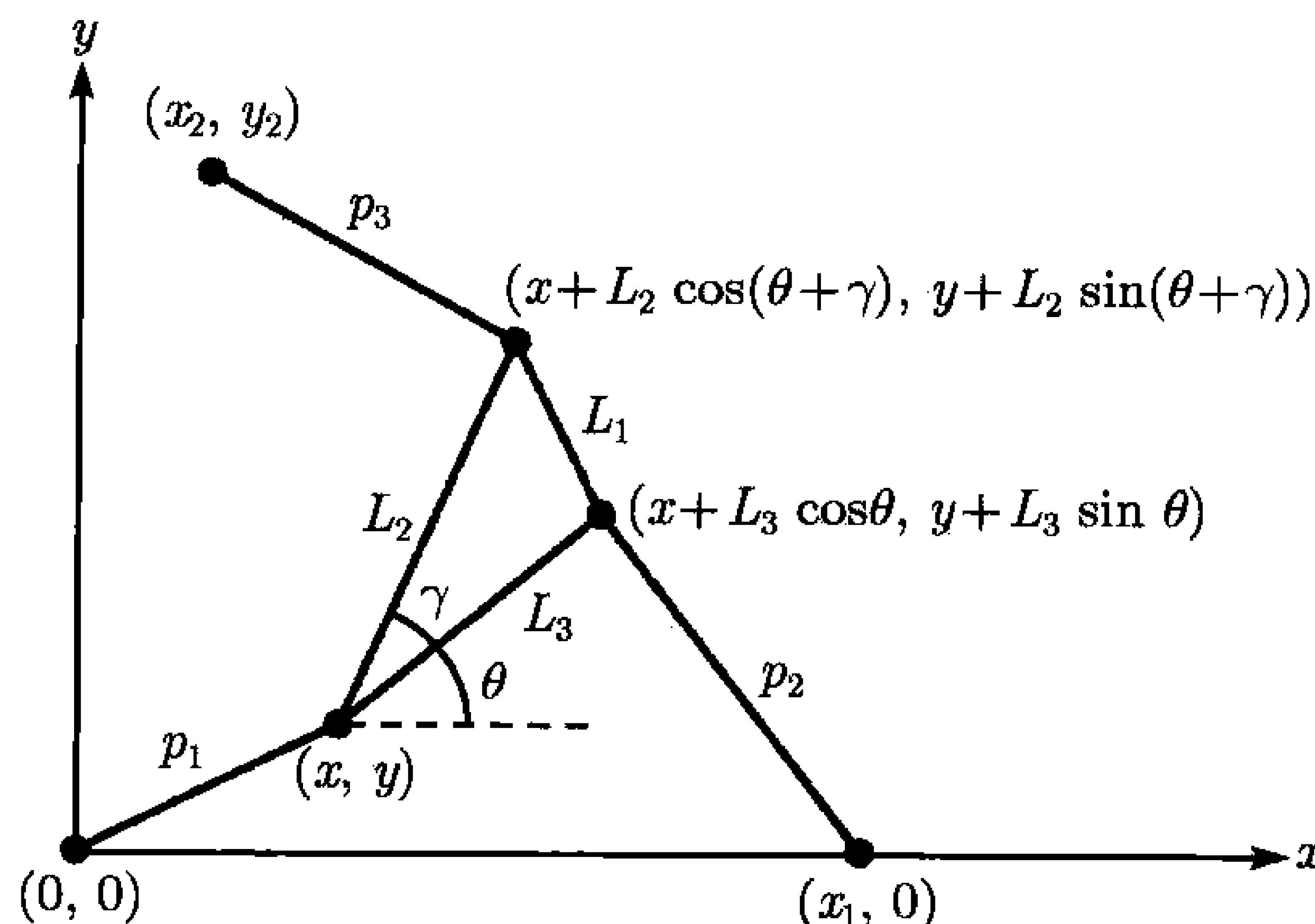


图 1-14 平面型 Stewart 平台示意图. 正向运动问题是用长度 p_1, p_2, p_3 确定未知量 x, y, θ

求给定 3 个支柱长度时的平台位置称为这个操纵装置的正向或定向运动问题. 即该问题是对每一组给定的 p_1, p_2, p_3 计算 (x, y) 及 θ. 因为有 3 个自由度, 我们自然期望用 3 个数去确定位置. 对于运动计划, 重要的是尽可能快地解这个问题 (经常是实时解决的). 不幸的是, 我们并不知道平面型 Stewart 平台正向运动问题的封闭形式的解.

目前几种最好的方法都要把图 1-14 中的几何图形简化为一个方程, 并且用本章介绍过的解法来解它. 你的任务是完成这个方程推导并写出执行其解的程序.

把简单的三角学知识用到图 1-14, 得到以下 3 个方程:

$$\begin{cases} p_1^2 = x^2 + y^2, \\ p_2^2 = (x+A_2)^2 + (y+B_2)^2, \\ p_3^2 = (x+A_3)^2 + (y+B_3)^2. \end{cases} \tag{1.38}$$

在这些方程中,

$$A_2 = L_3\cos\theta - x_1,$$

$$B_2 = L_3\sin\theta,$$

$$A_3 = L_2\cos(\theta+\gamma) - x_2 = L_2[\cos\theta\cos\gamma - \sin\theta\sin\gamma] - x_2,$$

$$B_3 = L_1\sin(\theta+\gamma) - y_2 = L_2[\cos\theta\sin\gamma + \sin\theta\cos\gamma] - y_2.$$

注意 (1.38) 是给定 x, y, θ, 求 p_1, p_2, p_3, 即是解平面型 Stewart 平台正向运动的反问题. 你的目标是解正向问题, 即对给定的 p_1, p_2, p_3 求 x, y, θ.

把 (1.38) 后两个方程展开, 再利用第一个方程得到

$$\begin{cases} p_2^2 = x^2 + y^2 + 2A_2x + 2B_2y + A_2^2 + B_2^2 = p_1^2 + 2A_2x + 2B_2y + A_2^2 + B_2^2, \\ p_3^2 = x^2 + y^2 + 2A_3x + 2B_3y + A_3^2 + B_3^2 = p_1^2 + 2A_3x + 2B_3y + A_3^2 + B_3^2, \end{cases}$$

它能够解出 x 和 y:

$$\begin{cases} x = \dfrac{N_1}{D} = \dfrac{B_3(p_2^2 - p_1^2 - A_2^2 - B_2^2) - B_2(p_3^2 - p_1^2 - A_3^2 - B_3^2)}{2(A_2B_3 - B_2A_3)}, \\ y = \dfrac{N_2}{D} = \dfrac{-A_3(p_2^2 - p_1^2 - A_2^2 - B_2^2) + A_2(p_3^2 - p_1^2 - A_3^2 - B_3^2)}{2(A_2B_3 - B_2A_3)}, \end{cases} \tag{1.39}$$

只要 $D = 2(A_2B_3 - B_2A_3) \neq 0$.

把这些 x 和 y 的表达式代入 (1.38) 中的第一个方程, 并乘以 D^2, 就得到一个方程, 即

$$f = N_1^2 + N_2^2 - p_1^2D^2 = 0 \tag{1.40}$$

只有单个未知量 θ. (回想一下, $p_1, p_2, p_3, L_1, L_2, L_3, \gamma, x_1, x_2, y_2$ 是已知的.) 如果能求出 $f(\theta)$ 的根, 那么相应的 x 和 y 的值立刻从 (1.39) 得到.

注意 $f(\theta)$ 是 $\sin\theta$ 和 $\cos\theta$ 的多项式, 所以给定任一根 θ, 对平台还有其他等价的根 $\theta+2\pi k$. 因此, 我们把 θ 限制在 $[-\pi, \pi]$ 内. 可以证明在这个区间内 $f(\theta)$ 最多有 6 个根.

建议习题

1. 对 $f(\theta)$ 写一个 MATLAB 函数文件. 参数 $L_1, L_2, L_3, \gamma, x_1, x_2, y_2$ 是固定的常数, 而支柱长度 p_1, p_2, p_3 对给定的位姿 (pose) 是已知的. 为了测试你的程序, 在图 1-15 中设置参数 $L_1 = 2, L_2 = L_3 = \sqrt{2}, \gamma = \frac{\pi}{2}, p_1 = p_2 = p_3 = \sqrt{5}$, 然后代入 $\theta = -\frac{\pi}{4}$ 或者 $\theta = \frac{\pi}{4}$, 分别对应于图 1-15a 和图 1-15b, 将使 $f(\theta) = 0$.
2. 画出 $f(\theta)$ 在 $[-\pi, \pi]$ 上的图形, 应该在 $\pm\frac{\pi}{4}$ 处有根.
3. 重新绘制图 1-15, MATLAB 命令

   ```
   >> plot ([u1 u2 u3 u1], [v1 v2 v3 v1], 'r'); hold on
   >> plot ([0 x1 x2], [0 0 y2], 'bo')
   ```

   将画出一个顶点是(u1, v1), (u2, r2), (u3, v3)的红色三角形, 并且在支柱的支撑点(0, 0), (0, x_1), (x_2, y_2)画出小圆. 另外, 画出支柱.
4. 求解由 $x_1 = 5, (x_2, y_2) = (0, 6), L_1 = L_3 = 3, L_2 = 3\sqrt{2}, \gamma = \frac{\pi}{4}, p_1 = p_2 = 5, p_3 = 3$ 所确定的平面型 Stewart 平台的正向运动问题. 从画出 $f(\theta)$ 的图形开始. 用第 1 章中的一种方程解法求出所有 4 种位姿, 并把它们画出来. 通过查对 p_1, p_2, p_3 是否是图中支柱的长度来验证答案.
5. 把支柱长度改为 $p_2 = 7$, 重新解这个问题, 对于这些参数, 有 6 种位姿.

6. 求支柱的一个长度 p_2, 其余的参数与第 4 步相同, 使其仅有两种位姿.
7. 计算出 p_2 的区间, 其余的参数与第 4 步相同, 使其分别有 0,2,4,6 种位姿.
8. 导出或查寻表示三维. 6 个自由度的 Stewart 平台的正向运动的方程. 写出 MATLAB 程序并演示用它来解正向运动问题. 关于棱柱形机器人手臂和平台的好的介绍可见 [4].

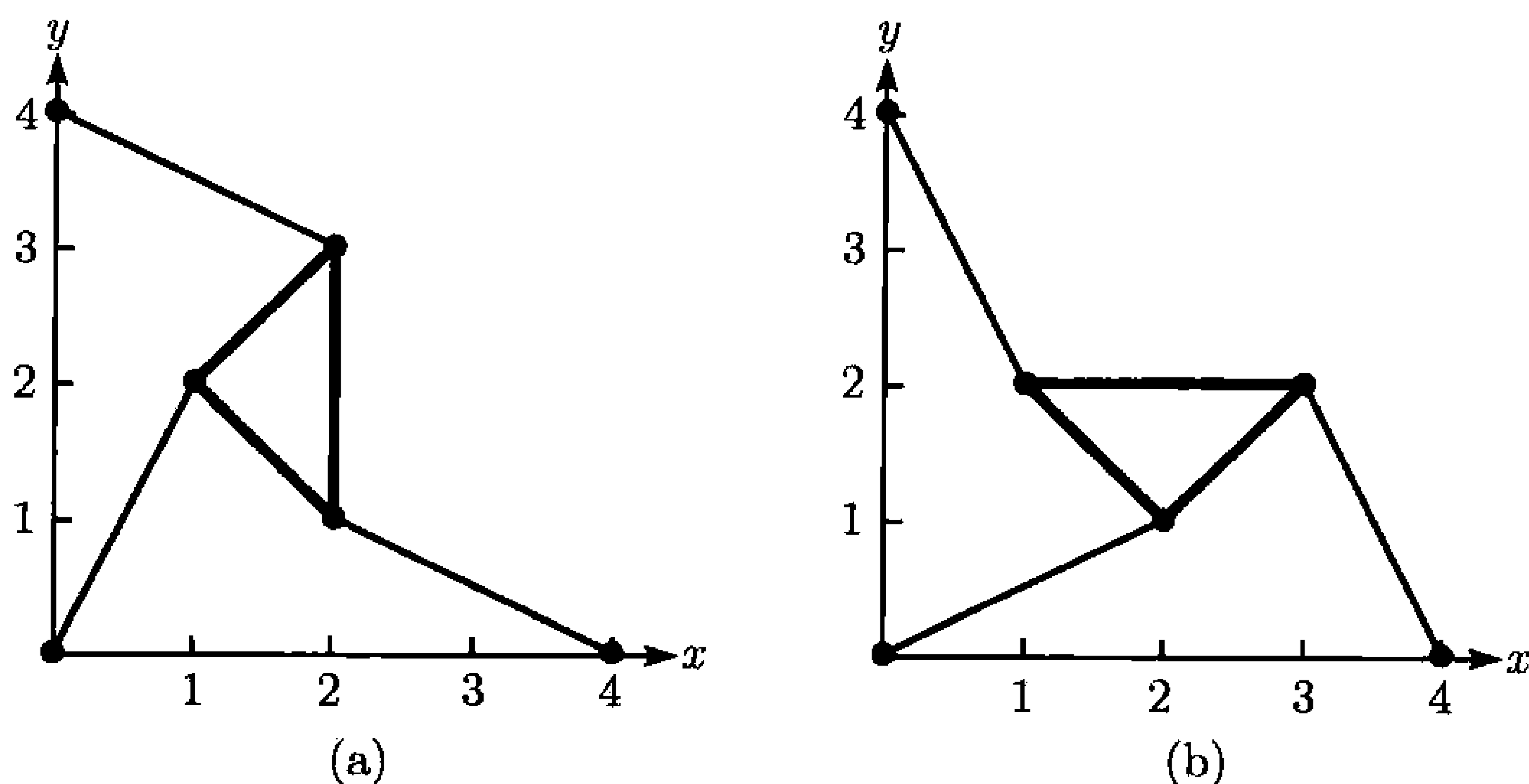


图 1-15 有相同臂长的平面型 Stewart 平台的两种位姿：每一种位姿对应于支柱长度 $p_1=p_2=p_3=\sqrt{5}$ 的 (1.38) 的一个解. 三角形的形状是由 $L_1=2, L_2=L_3=\sqrt{2}, \gamma=\frac{\pi}{2}$ 确定的

软件和进一步阅读

有许多确定非线性方程解的算法：像对分法这种虽然慢但总是收敛的算法; 与之对照的是收敛较快、但不能保证收敛的算法, 包括 Newton 法及其变形. 方程解法也可以分为两组, 取决于是否需要来自方程中的导数信息. 对分法、割线法及反二次插值法都是只需要一个提供给定输入的函数值的黑箱 (black box), 而 Newton 法需要导数. Brent 方法 [1] 兼取快慢两类算法之长, 并且不需要计算导数. 因此, 它广泛地用作一般用途的方程的解法, 而且收录在许多综合软件包里.

MATLAB 的`fzero`命令执行 Brent 方法仅需要一个初始区间或者一个初始估计作为输入. IMSL 的 ZBREN 程序、NAG 程序 c05adc 以及`netlib` FORTRAN 程序 fzero.f 都依靠这种基本方法. MATLAB 的`roots`命令用完全不同的方法求多项式的所有根, 计算伴随矩阵 (companion matrix) 的所有特征值, 这种伴随矩阵被构造为具有与这个多项式的所有根相等的特征根.

其他经常提到的算法都基于 Muller 方法和 Laguerre 方法, 它们在适当的条件下是立方收敛的. 更多的细节, 可查阅由 Householder[3]、Ostrowski[5] 及 Traub[6] 著的关于方程求解的经典教科书.

第2章　方　程　组

挠度的数学模型是结构工程师常会用到的基本工具. 在某负荷下结构性弯曲的程度依赖于材料的刚性, 这可用它的杨氏模数 (Young's modulus) 来度量. 压力与刚性之间的抗拒关系可由某一微分方程来模拟, 该方程经过离散化后可化为一线性方程组, 从而可求得其解.

为提高精度, 可使用更好的离散化方法, 这会使线性方程组的阶数变大且通常是稀疏的. 对于中等规模的矩阵, 高斯消去法是有效的; 而对大型、稀疏的系统, 需用特殊的迭代算法.

实例检验　2.5 节后的实例检验 2 研究了适用于固定梁和悬臂梁的 Euler-Bernoulli 模型的求解方法.

第 1 章讨论了求解只含一个变量的一个方程的方法. 本章考虑求解含有若干个变量的若干个联立方程的问题. 我们主要关注方程的个数与未知量的个数相等的情形.

高斯消去法是解适当规模线性方程组的主要方法. 本章首先建立这种著名方法的有效且稳定的形式, 接着转向大型方程组所需要的迭代方法, 最后建立求解非线性方程组的方法.

2.1　高斯消去法

考虑方程组

$$\begin{cases} x+y=3, \\ 3x-4y=2. \end{cases} \tag{2.1}$$

含有两个未知量、两个方程的方程组可从代数或几何的角度进行考虑. 从几何的观点来看, 每个线性方程代表平面上的一条直线, 如图 2-1 所示. 两条直线的交点 $x=2, y=1$ 满足这两个方程, 因此它就是我们要求的解.

几何观点对方程组解的形象化很有帮助, 但为了得到高精度的解, 我们还得回到代数. 高斯消去法是求解含 n 个变量的 n 个方程的有效方法. 在接下来的几节里, 我们将考察能最好地解决典型问题的高斯消去法的实现形式.

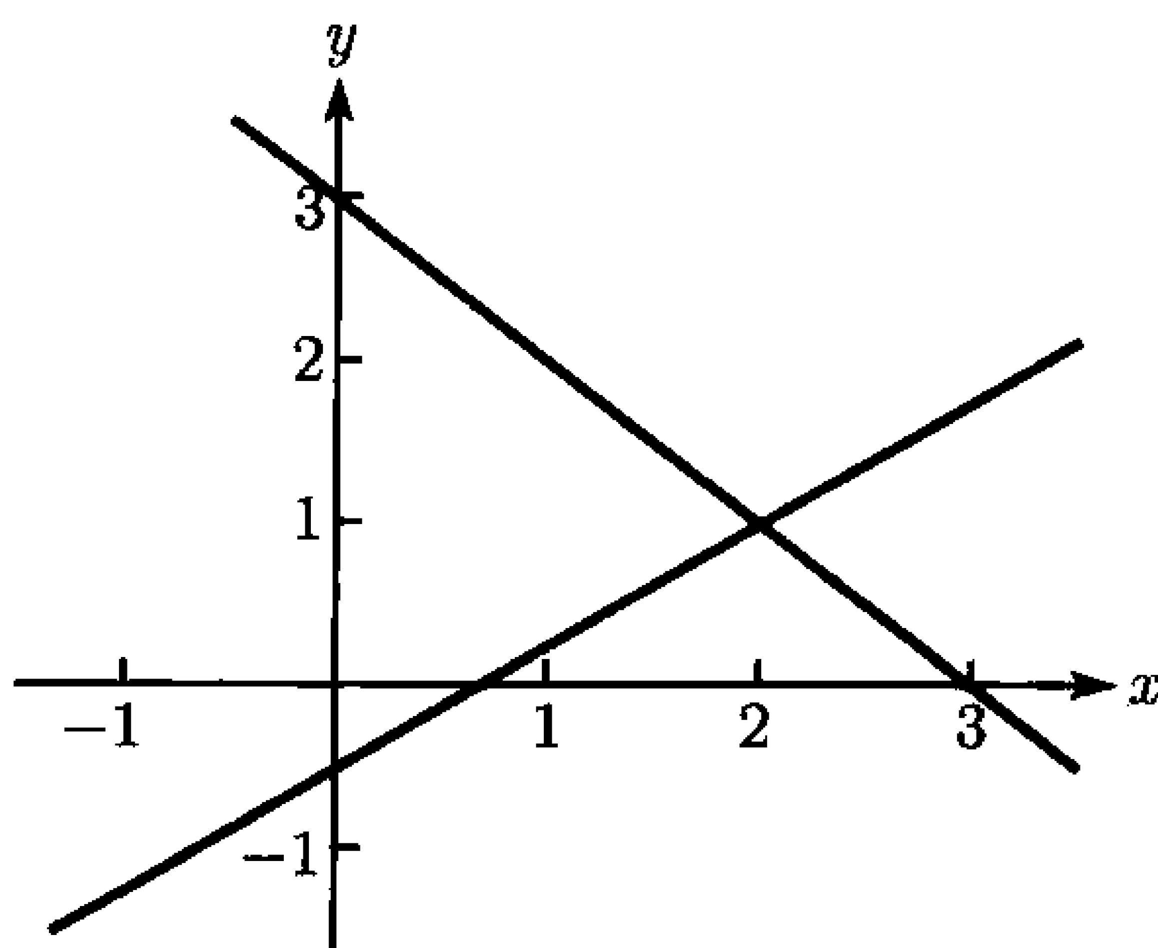


图 2-1 一个方程组的几何解: (2.1) 的每个方程代表平面上的一条直线, 相交的点即是解

2.1.1 基本的高斯消去法

我们首先描述高斯消去法的最简单形式. 事实上, 最简形式因太简单而不能保证执行到底, 更不用说求出精确解了. 所以这种基本方法需要加以改进, 其修正形式在后面几节予以介绍.

有 3 种有用的运算, 将它们作用于一个线性方程组可得到一个等价的方程组, 即两个方程组有相同的解. 这 3 种运算是

(1) 将一个方程与另一个方程进行交换;

(2) 将一个方程加上或减去另一方程的倍数;

(3) 将一个方程乘上一非零常数.

对于方程 (2.1), 可以用第 2 个方程减去第 1 个方程, 如此进行 3 次, 从第 2 个方程中消去变量 x. 从第 2 个方程减去 $3\cdot[x+y=3]$ 得到方程组

$$\begin{cases} x+y=3, \\ \quad -7y=-7. \end{cases} \tag{2.2}$$

从底下的一个方程开始, "回代求解" 可以得到一个完整的解, 即

$$-7y=-7\rightarrow y=1,$$

$$x+y=3\rightarrow x+(1)=3\rightarrow x=2.$$

则 (2.1) 的解是 $(x,y)=(2,1)$.

同样的消去过程可以这样完成: 不用写出变量, 把方程组写成所谓的表形式:

$$\left[\begin{array}{cc|c} 1 & 1 & 3 \\ 3 & -4 & 2 \end{array}\right] \rightarrow \begin{array}{c}\text{从第 2 行减} \\ \text{去}3\times\text{第1行}\end{array} \rightarrow \left[\begin{array}{cc|c} 1 & 1 & 3 \\ 0 & -7 & -7 \end{array}\right]. \tag{2.3}$$

这种表形式的优点在于: 当进行消去过程时, 所有的变量都隐藏起来了. 当表的左方的方阵化为 "三角形" 时, 我们就可以从底下的开始, 回代求出解.

例 2.1 以表形式应用高斯消去法求解含 3 个未知量的 3 个方程的方程组

$$\begin{cases} x+2y-z=3, \\ 2x+y-2z=3, \\ -3x+y+z=-6. \end{cases} \tag{2.4}$$

可以把它写成表的形式:

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 2 & 1 & -2 & 3 \\ -3 & 1 & 1 & -6 \end{array}\right]. \tag{2.5}$$

需要两步消去第 1 列:

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 2 & 1 & -2 & 3 \\ -3 & 1 & 1 & -6 \end{array}\right] \rightarrow \text{第 2 行减去} 2\times\text{第1行} \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ -3 & 1 & 1 & -6 \end{array}\right]$$

$$\rightarrow \text{第 3 行减去}(-3)\times\text{第1行} \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 7 & -2 & 3 \end{array}\right].$$

再需一步消去第 2 列:

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 7 & -2 & 3 \end{array}\right] \rightarrow \text{第 3 行减去}\left(-\tfrac{7}{3}\right)\times\text{第2行} \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 0 & -2 & -4 \end{array}\right].$$

回到方程组形式

$$\begin{cases} x+2y-z=3, \\ \qquad -3y=-3, \\ \qquad -2z=-4. \end{cases} \tag{2.6}$$

可以通过求解

$$\begin{cases} x=3-2y+z, \\ -3y=-3, \\ -2z=-4 \end{cases} \tag{2.7}$$

依次解出 z,y,x. 后一部分称为**回代**(back substitution) **或回解**(backsolving), 这是因为经消去过程后, 由底往上方程组易于解出. 解是 $x=3, y=1, z=2$. ◀

2.1.2 运算计数

本节对高斯消去法的两部分 (消去步骤和回代步骤) 作近似的运算 (次数) 计数. 为此, 把前面两例中执行的运算按一般的形式写出来是有帮助的. 首先回忆关于整数和的两个结论.

引理 2.1 对任意正整数 n, (a) $1+2+3+4+\cdots+n=n(n+1)/2$, (b) $1^2+2^2+3^2+4^2+\cdots+n^2=n(n+1)(2n+1)/6$.

含 n 个变量的 n 个方程的表的一般形式是

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right].$$

为执行消去步骤, 我们利用容许的行运算将下三角部分的元素都化为零.

可以把消去步骤写成循环

```
for j = 1 : n-1
  eliminate column j
end
```

其中, "**eliminate column j**" 的意思是 "使用行变换使主对角线下的每个元素即 $a_{j+1,j},a_{j+2,j},\cdots,a_{nj}$ 都为零". 例如, 对第 1 列执行消去, 我们需要把 $a_{21},\cdots,a_{n1}$ 都化为零. 这可以写成下列循环, 它包含在前面的循环中:

```
for j = 1 : n-1
  for i = j+1 : n
    eliminate entry a(i,j)
  end
end
```

这还需要利用将元素 a_{ij} 化为零的行运算完成双循环的内循环部分. 例如, 第一个要消去的元素是 a_{21}. 为此, 我们从第 2 行减去 $\frac{a_{21}}{a_{11}}$ 乘上第 1 行, 其中假设 $a_{11}\neq 0$. 即前两行从形式

$$\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \end{array}$$

变成了

$$\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a_{22}-\dfrac{a_{21}}{a_{11}}a_{12} & \cdots & a_{2n}-\dfrac{a_{21}}{a_{11}}a_{1n} & b_2-\dfrac{a_{21}}{a_{11}}b_1. \end{array}$$

计算运算次数, 这需要 1 次除法 (以得到乘子 $\frac{a_{21}}{a_{11}}$)、n 次乘法及 n 次加法. 用于消去第 1 列元素 a_{i1} 的行运算, 即

$$
\begin{array}{ccccc|c}
a_{11} & a_{12} & \cdots & a_{1n} & & b_1 \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & a_{i2}-\dfrac{a_{i1}}{a_{11}}a_{12} & \cdots & a_{in}-\dfrac{a_{i1}}{a_{11}}a_{1n} & & b_i-\dfrac{a_{i1}}{a_{11}}b_1
\end{array}
$$

需要相似的运算.

刚才描述的过程当数 a_{11} 非零时方可执行. 在高斯消去法中最终成为除数的这个数及其他数 a_{ii} 叫做**主元**(pivot). 一个零主元将导致算法终止, 正如目前我们所遇到的情况. 这种情形现在暂时忽略, 到 2.4 节时再予以谨慎的考虑.

回到运算计数, 注意到消去第 1 列时对每个元素 a_{i1} 用了 $n+1$ 次乘法/除法及 n 次加法/减法. 为简化计数工作, 我们将只记录乘法和除法的次数.

第 1 列消去后, 用同样的方法利用主元 a_{22} 消去第 2 列, 然后再依次消去其他列. 例如, 利用行运算消去元素 a_{ij} 为

$$
\begin{array}{cccccc|c}
0 & 0 & a_{jj} & a_{j,j+1} & \cdots & a_{jn} & b_j \\
\vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\
0 & 0 & 0 & a_{i,j+1}-\dfrac{a_{ij}}{a_{jj}}a_{j,j+1} & \cdots & a_{in}-\dfrac{a_{ij}}{a_{jj}}a_{jn} & b_i-\dfrac{a_{ij}}{a_{jj}}b_j.
\end{array}
$$

消去 a_{ij} 的行运算需要 $n-j+2$ 次乘法/除法 (按我们的记号, 比如说 a_{22}, 是指第 1 列消去后相应位置上已经修改了的数, 而非原来的 a_{22}).

将这个步骤插入到相同的双循环中得到

```
for j = 1 : n-1
  if abs(a(j,j))<eps; error('zero pivot encountered'); end
  for i = j+1 : n
    mult = a(i,j)/a(j,j);
    for k = j+1:n
      a(i,k) = a(i,k) - mult*a(j,k);
    end
    b(i) = b(i) - mult*b(j);
  end
end
```

对这段程序须作两点解释. 第一, 要求指标 k 从 j 移到 n, 会将 a_{ij} 位置处元素置为 0, 而从 $j+1$ 移到 n 是最有效的编码. 后者在 a_{ij} 处并不置为 0, 而它正是我们试图消去的! 虽然这似乎错了, 但注意到除去高斯消去过程或回代过程之外, 我们再也不会回到这个元素, 所以从效率的观点看, 在那儿置为零意味着浪费. 第二, 如果碰到一个零主元, 则使用 MATLAB 的`error`命令结束程序. 如前所述, 这种可能性

将在 2.4 节作更认真的讨论.

可以对高斯消去法消去过程中的运算 (或乘法/除法) 的次数作一总计数. 对 i 的每轮循环作了 $n-j+2$ 次运算, 且执行了 $n-j$ 轮, 则总数为 $(n-j+2)(n-j)$. 这个和须对 j 从 1 累加到 $n-1$, 这样的和按倒序可简单地写成

$$\begin{aligned} 3\times 1+4\times 2+\cdots+(n+1)(n-1) &= \sum_{l=1}^{n-1}(l+2)l = \sum_{l=1}^{n-1} l^2 + 2\sum_{l=1}^{n-1} l \\ &= \frac{(n-1)n(2n-1)}{6} + 2\frac{(n-1)n}{2} \\ &= \frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}, \end{aligned} \tag{2.8}$$

其中应用了引理 2.1.

高斯消去法消元步骤的运算计数 *对含有 n 个变量的 n 个方程的方程组, 完成消元步骤需 $\frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}$ 次乘法/除法.*

我们解释一下在统计高斯消去法的运算次数时忽略了减法次数的合理性. 当然可能会有人说这样做是否会丢失一些什么. 而事实上, 加法/减法的次数非常接近于乘法/除法的次数, 所以如果我们简单地把乘法/除法的次数加倍, 就将得到实际运算次数的一个很好估计 (练习 8 要求读者证明这一点). 在正常情况下, 我们仅对近似运算计数感兴趣, 因为在不同的计算机处理器上具体实现的细节是不同的. 要点在于, 按这种情形 (只计乘法/除法) 计得的总数是近似正比于算法执行的时间的.

消元过程完成后, 表成为上三角形

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{nn} & b_n \end{array}\right].$$

写成方程的形式, 即为

$$\left\{\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\ \ \vdots \\ a_{nn}x_n &= b_n, \end{aligned}\right. \tag{2.9}$$

其中 a_{ij} 是指已被修改而非原始的数据. 为完成求解 x 的计算, 必须执行回代步骤,

它是 (2.9) 的一个简单改写:

$$\begin{cases} x_1 = \frac{b_1 - a_{12}x_2 - \cdots - a_{1n}x_n}{a_{11}}, \\ x_2 = \frac{b_2 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}}, \\ \quad \vdots \\ x_n = \frac{b_n}{a_{nn}}. \end{cases} \tag{2.10}$$

由于方程组非零系数的三角形状, 我们从最下面一个方程开始, 一直求解到最上面一个方程. 按这种方法, 当计算下一个变量时, 所需要的那些 x_i 都已经知道了. 统计乘法/除法的次数, 我们得到

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2},$$

减法的次数是相似的 [实际上是 $\frac{(n-1)n}{2}$]. 用 MATLAB 语法格式表示回代过程即为

```
for i = n : -1 : 1
  for j = i+1 : n
    b(i) = b(i) - a(i,j)*x(j);
  end
  x(i) = b(i)/a(i,i);
end
```

高斯消去法回代过程的运算计数 对含 n 个变量的 n 个方程的三角形方程组, 完成回代步骤需要 $\frac{n^2}{2} + \frac{n}{2}$ 次乘法/除法.

把两部分的运算计数置于一起, 可以看到高斯消去法由不相等的两部分组成: 工作量相对多的消元步骤和工作量相对少的回代步骤. 如果我们忽略乘法/除法次数表达式中的低阶的项, 则可发现消元过程是 $\frac{n^3}{3}$ 阶的运算, 而回代过程是 $\frac{n^2}{2}$ 阶的.

我们将经常使用速写记号 O 表示 "是多少阶的", 则消元过程是 $O(\frac{n^3}{3})$ 的算法, 回代过程是 $O(\frac{n^2}{2})$ 的.

亮点　复杂性

运算计数表明, 利用高斯消去法对含 n 个未知量的 n 个方程的方程组求解的直接方法是 $O(\frac{n^3}{3})$ 的过程. 这对估计求解大型方程组所需的时间是有用的. 例如, 估计在一台特定的计算机上求解含 $n = 500$ 个方程的方程组所需要的时间, 我们可以通过求解一个含 $n = 50$ 个方程的方程组得到一个很好的猜测, 即对用掉的时间按比例扩大 $10^3 = 1\ 000$ 倍.

这种方法意味着重点在于大的 n, 其中幂次较低的 n 相比而言可以忽略不计. 例如, 若 $n = 100$, 回代步骤只占到高斯消去法计算时间的大约 1% 左右. 总体而言,

高斯消去法花费 $O(\frac{n^3}{3})+O(\frac{n^2}{2})$ 次运算. 按 O 的记法, 把 n 的不同次幂相加的结果是仅保留了最高次幂, 因为最高次幂决定了当 $n\to\infty$ 时的极限形态. 换而言之, 对大的 n, 复杂计数中的低阶项对算法的执行时间的估计没有太大的影响, 当仅需近似估计执行时间时可忽略不计.

例 2.2 对含 500 个变量的 500 个方程的方程组, 估计执行回代过程所需的时间, 所用计算机花费 1 秒钟完成消元.

因为我们已知消元所花时间比回代要多得多, 所以问题的答案将是 1 秒钟的一个分数. 消元步骤的乘法/除法运算次数的近似数是 $(500)^3/3$, 回代步骤是 $(500)^2/2$, 则我们估计回代过程所需的时间为

$$\frac{(500)^2/2}{(500)^3/3}=\frac{3}{500\times 2}=0.003\text{s}.$$ ◀

这个例子说明两点: (1) 运算计数中 n 的较低次幂可放心地忽略不计; (2) 高斯消去法的两部分在执行时间上可以很不一样——总的计算时间是 1.003 秒, 其大部分花在了消元步骤上. 下一个例子说明了第三点. 尽管回代的时间有时可以忽略不计, 但也可以用它来估计一些重要的运算.

例 2.3 在一台特定的计算机上, 执行一个 500×500 的三角形矩阵的回代过程需要 0.1 秒. 用高斯消去法求解一个含 300 个未知量的 300 个方程的一般方程组, 估计所需花费的时间.

计算机在 0.1 秒内可执行 $(500)^2/2$ 次乘法/除法, 即运算速度为 $(500)^2(10)/2=1.25\times 10^6$ 次运算/秒. 求解这个一般 (非奇异的) 方程组需要大约 $(300)^3/3$ 次运算, 在大约

$$\frac{(300)^3/3}{(500)^2\times 10/2}\approx 7.2\text{s}.$$

内可以完成. ◀

习题 2.1

1. 用高斯消去法求解下列方程组:

(a) $\begin{cases}2x-3y=2,\\5x-6y=8;\end{cases}$ (b) $\begin{cases}x+2y=-1,\\2x+3y=1;\end{cases}$ (c) $\begin{cases}-x+y=2,\\3x+4y=15.\end{cases}$

2. 用高斯消去法求解下列方程组:

(a) $\begin{cases}2x-2y-z=-2,\\4x+y-2z=1,\\-2x+y-z=-3;\end{cases}$ (b) $\begin{cases}x+2y-z=2,\\3y+z=4,\\2x-y+z=2;\end{cases}$ (c) $\begin{cases}2x+y-4z=-7,\\x-y+z=-2,\\-x+3y-2z=6.\end{cases}$

3. 用回代法求解：

(a) $\begin{cases} 3x - 4y + 5z = 2, \\ \quad\quad 3y - 4z = -1, \\ \quad\quad\quad\quad 5z = 5; \end{cases}$ (b) $\begin{cases} x - 2y + z = 2, \\ \quad 4y - 3z = 1, \\ \quad\quad -3z = 3. \end{cases}$

4. 求解表形式：

(a) $\left[\begin{array}{ccc|c} 3 & -4 & -2 & 3 \\ 6 & -6 & 1 & 2 \\ -3 & 8 & 2 & -1 \end{array}\right]$； (b) $\left[\begin{array}{ccc|c} 2 & 1 & -1 & 2 \\ 6 & 2 & -2 & 8 \\ 4 & 6 & -3 & 5 \end{array}\right]$.

5. 利用高斯消去法的近似运算计数 $\frac{n^3}{3}$, 若 n 增大 3 倍, 估计求解含 n 个未知量的 n 个方程的方程组要花多长时间.

6. 假设你的计算机要在 0.5 秒内完成 1 000 个方程的回代过程. 利用回代过程的近似运算计数 $\frac{n^2}{2}$ 及消元过程的 $\frac{n^3}{3}$ 次, 估计对上述规模的方程组完成高斯消去法需要多长时间. 将你的答案四舍五入到最近的秒数.

7. 假设所给的计算机对一个 400×400 的上三角矩阵方程完成回代过程需要 0.2 秒. 估计求解一个含 900 个未知量的 900 个方程的一般方程组所需的时间. 将你的答案四舍五入到最近的分钟数.

8. 对高斯消去法的 (a) 消元步骤、(b) 回代步骤找出准确的加法/减法次数. 对大的 n, 这些运算计数跟乘法/除法的运算计数相比会相差多少?

计算机问题 2.1

1. 把本节的程序片段放在一起形成 "基本" 高斯消去法的 MATLAB 程序 (意即不允许进行交换). 利用此程序求解习题 2 的方程组.

2. 设 $\boldsymbol{H}$ 表示 $n \times n$ 的 Hilbert 矩阵, 其 (i, j) 元素是 $\frac{1}{i+j-1}$. 利用计算机问题 1 的 MATLAB 程序求解 $\boldsymbol{Hx} = \boldsymbol{b}$, 其中 $\boldsymbol{b}$ 分别是 (a) $n = 2$; (b) $n = 5$; (c) $n = 10$ 的其列元素全为 1 的向量.

2.2 *LU*分解

通过进一步利用表形式, 我们可以把方程组表示成矩阵形式. 矩阵形式由于简化了算法及其分析, 从而最终可以节约时间.

2.2.1 高斯消去法的矩阵形式

方程组 (2.1) 可以写成 $\boldsymbol{Ax} = \boldsymbol{b}$ 的矩阵形式或

$$\begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}. \tag{2.11}$$

我们通常记**系数矩阵**(coefficient matrix) 为 $\boldsymbol{A}$, **右端项**(right-hand side) 向量为 $\boldsymbol{b}$, 在方程组的矩阵形式中, 我们把 $\boldsymbol{x}$ 看作是列向量, $\boldsymbol{Ax}$ 看作是矩阵与向量的乘积. 我

们想找到 $\boldsymbol{x}$ 使得向量 $\boldsymbol{Ax}$ 等于向量 $\boldsymbol{b}$. 当然这相当于 $\boldsymbol{Ax}$ 与 $\boldsymbol{b}$ 所有的分量对应相等, 而这正是原始方程组 (2.1) 所需要的.

把方程组写成矩阵形式的优势在于, 我们可以利用像矩阵相乘这样的矩阵运算以追踪高斯消去法的步骤. $\boldsymbol{LU}$ 分解是高斯消去法的矩阵表示, 它把系数矩阵 $\boldsymbol{A}$ 写成一个**下三角矩阵** $\boldsymbol{L}$ 和一个**上三角矩阵** $\boldsymbol{U}$ 的乘积. $\boldsymbol{LU}$ 分解是高斯消去法的一种表示形式, 而这是科学与工程中具有悠久传统的一种表示方式, 即把一个复杂的对象分解成简单的部分.

定义 2.2 一个 $m \times n$ 矩阵 $\boldsymbol{L}$ 是**下三角的**(lower triangular), 若其元素满足 $l_{ij} = 0$(当 $i < j$ 时). 一个 $m \times n$ 矩阵 $\boldsymbol{U}$ 是**上三角的**(upper triangular), 若其元素满足 $u_{ij} = 0$(当 $i > j$ 时).

例 2.4 求出 (2.11) 中矩阵 $\boldsymbol{A}$ 的 $\boldsymbol{LU}$ 分解.

消元步骤与早先见到的表形式是一样的:

$$\begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \to \begin{array}{c} \text{从第 2 行} \\ \text{减去} 3 \times \text{第1行} \end{array} \to \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} = \boldsymbol{U}. \tag{2.12}$$

差别在于现在我们存储了消元步骤中用到的乘数 3. 注意到我们已定义 $\boldsymbol{U}$ 是表示高斯消去法结果的上三角矩阵. 定义 $\boldsymbol{L}$ 为主对角元为 1 的 2×2 的下三角矩阵, 乘数 3 在 (2,1) 位置上:

$$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}.$$

然后检验有

$$\boldsymbol{LU} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} = \boldsymbol{A}. \tag{2.13}$$

◀

不久我们会讨论这样做的理由, 但这里先用一个 3×3 的例子来说明这些步骤的使用.

例 2.5 求

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix}. \tag{2.14}$$

的 $\boldsymbol{LU}$ 分解.

这个矩阵是方程组 (2.4) 的系数矩阵, 消元步骤如前进行:

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} \xrightarrow[\text{减去 } 2\times\text{第 1 行}]{\text{第 2 行}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ -3 & 1 & 1 \end{bmatrix}$$

$$\xrightarrow[(-3)\times\text{第 1 行}]{\text{第 3 行减去}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 7 & -2 \end{bmatrix} \xrightarrow[\left(-\frac{7}{3}\right)\times\text{第 2 行}]{\text{第 3 行减去}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} = \boldsymbol{U}.$$

如前例一样构造下三角阵 $\boldsymbol{L}$: 主对角线元素都置为 1, 乘数置于在消元中用到它们的特定的地方下三角处, 即

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix}. \tag{2.15}$$

注意, 例如, 由于 2 是用于消去 $\boldsymbol{A}$ 的 (2, 1) 位置的元素, 所以 2 置于 $\boldsymbol{L}$ 的 (2, 1) 位置处. 现在检验有

$$\boldsymbol{LU} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} = \boldsymbol{A}. \tag{2.16}$$

◀

这个过程能给出 $\boldsymbol{LU}$ 分解的理由是基于下面 3 个关于下三角矩阵的事实.

事实 1 *设 $\boldsymbol{L}_{ij}(-c)$ 表示主对角元为 $1, (i,j)$ 位置为 $-c$, 其余元素都为 0 的下三角矩阵. 则 $\boldsymbol{A} \to \boldsymbol{L}_{ij}(-c)\boldsymbol{A}$ 表示行运算 "从 i 行减去 c 乘 j 行".*

例如, 用 $\boldsymbol{L}_{21}(-c)$ 相乘得到

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} - ca_{11} & a_{22} - ca_{12} & a_{23} - ca_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

事实 2 $\boldsymbol{L}_{ij}(-c)^{-1} = \boldsymbol{L}_{ij}(c)$.

例如

$$\begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

利用事实 1 和事实 2, 我们能够理解例 2.4 的 $\boldsymbol{LU}$ 分解. 由于消元步骤能表示成

$$\boldsymbol{L}_{21}(-3)\boldsymbol{A}=\begin{bmatrix}1&0\\-3&1\end{bmatrix}\begin{bmatrix}1&1\\3&-4\end{bmatrix}=\begin{bmatrix}1&1\\0&-7\end{bmatrix},$$

我们在式子两边同时乘上 $\boldsymbol{L}_{21}(-3)^{-1}$ 得到

$$\boldsymbol{A}=\begin{bmatrix}1&1\\3&-4\end{bmatrix}=\begin{bmatrix}1&0\\3&1\end{bmatrix}\begin{bmatrix}1&1\\0&-7\end{bmatrix},$$

而这正是 $\boldsymbol{A}$ 的 $\boldsymbol{LU}$ 分解.

为处理 $n>2$ 的 $n\times n$ 的矩阵, 我们还需要一个事实.

事实 3 下面的矩阵乘积方程成立:

$$\begin{bmatrix}1&&\\c_1&1&\\&&1\end{bmatrix}\begin{bmatrix}1&&\\&1&\\c_2&&1\end{bmatrix}\begin{bmatrix}1&&\\&1&\\&c_3&1\end{bmatrix}=\begin{bmatrix}1&&\\c_1&1&\\c_2&c_3&1\end{bmatrix}.$$

利用这个事实, 我们可以把所有 $\boldsymbol{L}_{ij}$ 的逆结合起来写成一个矩阵, 即 $\boldsymbol{LU}$ 分解的矩阵 $\boldsymbol{L}$. 对于例 2.5, 这相当于

$$\begin{bmatrix}1&&\\&1&\\&\frac{7}{3}&1\end{bmatrix}\begin{bmatrix}1&&\\&1&\\3&&1\end{bmatrix}\begin{bmatrix}1&&\\-2&1&\\&&1\end{bmatrix}\begin{bmatrix}1&2&-1\\2&1&-2\\-3&1&1\end{bmatrix}=\begin{bmatrix}1&2&-1\\0&-3&0\\0&0&-2\end{bmatrix}=\boldsymbol{U}$$

$$\boldsymbol{A}=\begin{bmatrix}1&&\\2&1&\\&&1\end{bmatrix}\begin{bmatrix}1&&\\&1&\\-3&&1\end{bmatrix}\begin{bmatrix}1&&\\&1&\\&-\frac{7}{3}&1\end{bmatrix}\begin{bmatrix}1&2&-1\\0&-3&0\\0&0&-2\end{bmatrix}$$

$$=\begin{bmatrix}1&&\\2&1&\\-3&-\frac{7}{3}&1\end{bmatrix}\begin{bmatrix}1&2&-1\\0&-3&0\\0&0&-2\end{bmatrix}=\boldsymbol{LU}. \tag{2.17}$$

2.2.2 利用 $\boldsymbol{LU}$ 分解的回代过程

既然我们已经把高斯消去法的消元步骤表示成一个矩阵乘积 $\boldsymbol{LU}$ 的形式, 那如何来表示回代步骤呢? 更重要的是, 实际上我们如何才能得到解 $\boldsymbol{x}$?

一旦得到了 $\boldsymbol{L}$ 和 $\boldsymbol{U}$, 则问题 $\boldsymbol{Ax}=\boldsymbol{b}$ 可以写成 $\boldsymbol{LUx}=\boldsymbol{b}$. 定义一个新的 “辅助” 向量 $\boldsymbol{c}=\boldsymbol{Ux}$, 则回代是一个两步的过程:

(a) 解 $\boldsymbol{Lc}=\boldsymbol{b}$ 得到 $\boldsymbol{c}$;

(b) 解 $\boldsymbol{Ux}=\boldsymbol{c}$ 得到 $\boldsymbol{x}$.

由于 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 是三角形矩阵, 上面两步都是简明易解的. 我们用以前的两个例子进行验证.

例 2.6 利用 $\boldsymbol{LU}$ 分解 (2.13) 求解方程组 (2.11).

方程组有 $\boldsymbol{LU}$ 分解

$$\begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} = \boldsymbol{LU} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix},$$

由 (2.13) 及右端项 $\boldsymbol{b}=[3,2]$, 步骤 (a) 是

$$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix},$$

这对应于方程组

$$\begin{cases} c_1 + 0c_2 = 3, \\ 3c_1 + c_2 = 2. \end{cases}$$

从上面一个方程开始求解, 解是 $c_1=3, c_2=-7$.

步骤 (b) 是

$$\begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -7 \end{bmatrix},$$

这对应于方程组

$$\begin{cases} x_1 + x_2 = 3, \\ -7x_2 = -7. \end{cases}$$

从下面一个方程开始求解, 得到解为 $x_2=1, x_1=2$. 这跟前面用 "经典" 的高斯消去法计算得到的结果是一致的. ◀

例 2.7 利用 $\boldsymbol{LU}$ 分解 (2.16) 求解方程组 (2.4).

方程组有 $\boldsymbol{LU}$ 分解

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} = \boldsymbol{LU} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix}.$$

由 (2.16) 及 $\boldsymbol{b}=(3,3,-6)$, 步骤 $\boldsymbol{Lc}=\boldsymbol{b}$ 是

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ -6 \end{bmatrix},$$

这对应于方程组

$$\begin{cases} c_1 = 3, \\ 2c_1 + c_2 = 3, \\ -3c_1 - \frac{7}{3}c_2 + c_3 = -6. \end{cases}$$

从上面第一个方程起, 求得解为 $c_1 = 3, c_2 = -3, c_3 = -4$.

步骤 $\boldsymbol{Ux} = \boldsymbol{c}$ 是

$$\begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -3 \\ -4 \end{bmatrix},$$

这对应于方程组

$$\begin{cases} x_1 + 2x_2 - x_3 = 3, \\ -3x_2 = -3, \\ -2x_3 = -4, \end{cases}$$

从下往上求解得到 $\boldsymbol{x} = [3, 1, 2]$. ◀

2.2.3 *LU* 分解的复杂性

既然我们已经知道了怎么进行 $\boldsymbol{LU}$ 分解, 这里再说一下为什么要进行 $\boldsymbol{LU}$ 分解: 经典的高斯消去法在计算过程中的消元步骤中既要处理 $\boldsymbol{A}$, 又要处理 $\boldsymbol{b}$, 而正如我们所见, 这是整个过程中花费工作量最多的部分. 现在若假设我们要求解若干个不同的问题, 它们有相同的 $\boldsymbol{A}$ 但 $\boldsymbol{b}$ 不同, 即给了我们一批问题

$$\begin{gathered} \boldsymbol{Ax} = \boldsymbol{b}_1, \\ \boldsymbol{Ax} = \boldsymbol{b}_2, \\ \vdots \\ \boldsymbol{Ax} = \boldsymbol{b}_k, \end{gathered}$$

其中右端向量为不同的 $\boldsymbol{b}_i$. 因为我们必须对每一个问题从头到尾进行求解, 所以当 $\boldsymbol{A}$ 为 $n \times n$ 的矩阵时, 经典的高斯消去法将需要大约 $\frac{kn^3}{3}$ 次运算. 而另一方面, 借助于 $\boldsymbol{LU}$ 分解, 直到消元过程 ($\boldsymbol{A} = \boldsymbol{LU}$ 分解) 结束, 我们的计算才用到右端项 $\boldsymbol{b}$. 依靠这样将 $\boldsymbol{A}$ 与 $\boldsymbol{b}$ 隔离开来的计算, 求解前面的一批方程仅用到一次消元, 以及对每一个新的 $\boldsymbol{b}$ 再进行的两次回代 ($\boldsymbol{Lc} = \boldsymbol{b}, \boldsymbol{Ux} = \boldsymbol{c}$). 因此, 借助 $\boldsymbol{LU}$ 分解, 运算的次数大约是 $\frac{n^3}{3} + kn^2$. 当 n^2 比 n^3 小 (即当 n 比较大) 时, 这个差别就显示出 $\boldsymbol{LU}$ 分解的意义了.

亮点 复杂性

对高斯消去法应用 $\boldsymbol{LU}$ 分解方法, 主要是由于广泛存在具有 $\boldsymbol{Ax}=\boldsymbol{b}_1, \boldsymbol{Ax}=\boldsymbol{b}_2, \cdots$ 这样形式的问题. 通常, $\boldsymbol{A}$ 是所谓的结构矩阵, 它仅依赖于一个力学系统或动力系统的设计, $\boldsymbol{b}$ 则对应于一个 "负载向量" (loading vector). 在结构工程学中, 负载向量给出了结构上不同点处的作用力. 解 $\boldsymbol{x}$ 则对应于由负载的特定组合所诱导出的结构上的应力. 对不同的 $\boldsymbol{b}$ 求出的 $\boldsymbol{Ax}=\boldsymbol{b}$ 的解将被用于测试潜在的结构设计. 实例检验 2 对横梁的负载进行了这种分析.

即使当 $k=1$ 时, 相比于经典的高斯消去法, 利用 $\boldsymbol{A}=\boldsymbol{LU}$ 分解的方法也没有额外的计算工作量. 尽管 $\boldsymbol{LU}$ 分解法好像有经典的高斯消去法中所没有的一个额外的回代, 其实这只是替代了由于缺少右端项 $\boldsymbol{b}$ 而在消元过程中节省下来的那些计算.

如果一开始时就已知所有的 $\boldsymbol{b}_i$, 我们就可以用相同的运算次数同时求解所有的 k 个问题. 但在典型的应用问题中, 要求我们求解的是一些 $\boldsymbol{Ax}=\boldsymbol{b}_i$ 问题而其他的 $\boldsymbol{b}_i$ 尚未知道. 对有相同系数矩阵 $\boldsymbol{A}$ 的现有的与将有的问题, $\boldsymbol{LU}$ 分解给出了有效的处理方法.

例 2.8 假设将一个 300×300 的矩阵分解成 $\boldsymbol{A}=\boldsymbol{LU}$ 要花 1 秒钟, 则在下一个 1 秒钟内可以解多少个问题 $\boldsymbol{Ax}=\boldsymbol{b}_1,\cdots,\boldsymbol{Ax}=\boldsymbol{b}_k$?

对每个 $\boldsymbol{b}_i$ 执行的两次回代总共需要 $\frac{2n^2}{2}=n^2$ 次运算. 所以可以处理的 $\boldsymbol{b}_i$ 的近似数目为

$$\frac{\frac{n^3}{3}}{n^2}=\frac{n}{3}=100. \qquad ◀$$

在我们对有效地运用高斯消去法的探讨中, $\boldsymbol{LU}$ 分解将是一个有意义的步骤. 可遗憾的是, 不是每个矩阵都有这样的分解.

例 2.9 证明 $\boldsymbol{A}=\begin{bmatrix}0&1\\1&1\end{bmatrix}$ 没有 $\boldsymbol{LU}$ 分解.

分解必须有形式

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}=\begin{bmatrix}1&0\\a&1\end{bmatrix}\begin{bmatrix}b&c\\0&d\end{bmatrix}=\begin{bmatrix}b&c\\ab&ac+d\end{bmatrix},$$

使两边系数对应相等得到 $b=0$ 及 $ab=1$, 矛盾. ◀

并非所有的矩阵都有 $\boldsymbol{LU}$ 分解, 这一事实意味着在我们得到一个利用 $\boldsymbol{LU}$ 分解的一般的高斯消去算法之前, 尚有更多的工作要做. 2.3 节将描述摆动的相关问题. 2.4 节将介绍 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解, 它会克服这两个问题.

习题 2.2

1. 求出所给矩阵的 LU 分解, 并用矩阵相乘检验:

(a) $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}$; (c) $\begin{bmatrix} 3 & -4 \\ -5 & 2 \end{bmatrix}$.

2. 求出所给矩阵的 LU 分解, 并用矩阵相乘检验:

(a) $\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{bmatrix}$; (b) $\begin{bmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & -1 & 1 & 2 \\ 0 & 2 & 1 & 0 \\ 1 & 3 & 4 & 4 \\ 0 & 2 & 1 & -1 \end{bmatrix}$.

3. 通过求出 LU 分解求解方程组, 执行两步回代过程:

(a) $\begin{bmatrix} 3 & 7 \\ 6 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -11 \end{bmatrix}$; (b) $\begin{bmatrix} 2 & 3 \\ 4 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$.

4. 通过求出 LU 分解求解方程组, 执行两步回代过程:

(a) $\begin{bmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$; (b) $\begin{bmatrix} 4 & 2 & 0 \\ 4 & 4 & 2 \\ 2 & 2 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$.

5. 解方程 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 4 & 1 & 2 & 1 \end{bmatrix}\begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \end{bmatrix}.$$

6. 对所给的 100×100 的矩阵 $\boldsymbol{A}$, 利用 $\boldsymbol{A} = \boldsymbol{LU}$ 分解方法, 你的计算机恰好在 1 分钟内可以解出 50 个问题 $\boldsymbol{Ax} = \boldsymbol{b}_1, \cdots, \boldsymbol{Ax} = \boldsymbol{b}_{50}$, 那么计算机花在 $\boldsymbol{A} = \boldsymbol{LU}$ 分解上的时间是多少分钟? 四舍五入你的结果到最近的秒数.

7. 假设你的计算机每秒钟可以解出 100 个 $\boldsymbol{Ux} = \boldsymbol{c}$ 这种类型的问题, 其中 $\boldsymbol{U}$ 是一个 50×50 的上三角矩阵. 估计求解一个满的 500×500 的矩阵问题 $\boldsymbol{Ax} = \boldsymbol{b}$ 需要花多长时间. 以分钟数和秒数回答.

8. 假设你的计算机 1 秒钟可以解出一个 200×200 的线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$. 利用 LU 分解方法, 求解 4 个有相同系数矩阵的含 500 个未知量的 500 个方程的方程组, 估计需要的时间 (到最近的秒).

9. 设 $\boldsymbol{A}$ 为 $n \times n$ 矩阵, 假设你的计算机利用 LU 分解求 100 个问题 $\boldsymbol{Ax} = \boldsymbol{b}_1, \cdots, \boldsymbol{Ax} = \boldsymbol{b}_{100}$ 与求解第一个问题 $\boldsymbol{Ax} = \boldsymbol{b}_0$ 花费同样多的时间, 试估计 n.

计算机问题 2.2

1. 利用 2.1 节里的高斯消元的程序片段, 写一段 MATLAB 程序: 把矩阵作为输入, L 和 U 为输出. 不允许有行交换: 程序的设计要使得碰到零主元时就终止. 利用分解习题 2 中的矩阵来检验你的程序.

2. 在计算机问题 1 中, 为你的程序段里加入两步回代过程, 并利用它求解习题 4 的方程组.

2.3 误差的来源

正如我们迄今所述, 高斯消去法有两个主要的误差来源. 病态性 (ill-conditioning) 是解对输入数据的敏感性. 我们将利用第 1 章的后向误差和前向误差的概念来讨论条件数. 由于在计算病态性矩阵方程的解时误差几乎是不可避免的, 所以在可能的时候识别并避免病态性的矩阵是重要的. 误差的第二个来源是摆动, 这在绝大多数问题中利用一个叫做部分主元的简单修正方法就可予以避免, 而这正是 2.4 节的主题.

接下来将介绍向量和矩阵范数的概念, 它们用以度量误差的大小, 而目前用的是向量. 我们将主要介绍所谓的无穷大范数.

2.3.1 误差放大及条件数

在第 1 章中, 我们发现一些方程求解问题的后向误差与前向误差有很大的差异. 对线性方程组而言也是如此. 为量化误差的大小, 我们先给出向量的无穷大范数的定义.

定义 2.3 向量 $\boldsymbol{x} = (x_1, \cdots, x_n)$ 的**无穷大范数**(infinity norm) 或**最大范数**(maximum norm) 是 $\|\boldsymbol{x}\|_\infty = \max|x_i|, i = 1, 2, \cdots, n$. 即 $\boldsymbol{x}$ 的分量绝对值中的最大值.

后向误差与前向误差可与定义 1.8 类似地予以定义. 后向误差代表输入或问题数据方面的差, 前向误差代表输出或算法解方面的差.

定义 2.4 设 $\boldsymbol{x}_c$ 为线性方程组 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ 的近似解, 则**残差**(residual) 是向量 $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_c$. **后向误差**是残差范数 $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_c\|_\infty$, **前向误差**是 $\|\boldsymbol{x} - \boldsymbol{x}_c\|_\infty$.

例 2.10 求出方程组

$$\begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

的近似解 $x_c = [1, 1]$ 的后向误差与前向误差.

正确的解是 $\boldsymbol{x} = [2, 1]$. 按无穷大范数, 后向误差为

$$\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_c\|_\infty = \left\| \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 1 \\ 3 \end{bmatrix} \right\|_\infty = 3$$

前向误差为

$$\|\boldsymbol{x} - \boldsymbol{x}_c\|_\infty = \left\| \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|_\infty = 1.$$

◀

在其他情形下, 后向误差与前向误差会有不同的数量级.

例 2.11 求出方程组

$$\begin{cases} x_1 + x_2 = 2, \\ 1.000\ 1x_1 + x_2 = 2.000\ 1 \end{cases} \tag{2.18}$$

近似解 $[-1, 3.000\ 1]$ 的前向误差与后向误差.

首先, 求出精确解 $[x_1, x_2]$. 高斯消去法由下列步骤组成:

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1.000\ 1 & 1 & 2.000\ 1 \end{array}\right] \rightarrow \begin{array}{c}\text{第2行减去} \\ 1.000\ 1\times\text{第 1行}\end{array} \rightarrow \left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & -0.000\ 1 & -0.000\ 1 \end{array}\right]$$

求解相应的方程组

$$\begin{cases} x_1 + x_2 = 2, \\ -0.000\ 1x_2 = -0.000\ 1, \end{cases}$$

得到解 $[x_1, x_2] = [1, 1]$.

后向误差是向量

$$\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_c = \begin{bmatrix} 2 \\ 2.000\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1.000\ 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 3.000\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2.000\ 1 \end{bmatrix} - \begin{bmatrix} 2.000\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.000\ 1 \\ 0.000\ 1 \end{bmatrix}$$

的无穷大范数, 为 0.000 1. 前向误差是差

$$\boldsymbol{x} - \boldsymbol{x}_c = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 3.000\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -2.000\ 1 \end{bmatrix}$$

的无穷大范数, 为 2.000 1. ◀

图 2-2 表明: 可以同时有一个小的后向误差与一个大的前向误差. 尽管 "近似根"$(-1, 3.000\ 1)$ 离精确根 $(1, 1)$ 相对较远, 但它几乎在两条直线上. 由于这两条直线几乎平行, 因此这是可能的. 如果这两条直线根本不平行, 则前向误差与后向误差在数量上将会很接近.

记残差为 $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_c$, 则方程组 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ 的**相对后向误差**定义为

$$\frac{\|\boldsymbol{r}\|_\infty}{\|\boldsymbol{b}\|_\infty},$$

相对前向误差定义为

$$\frac{\|\boldsymbol{x} - \boldsymbol{x}_c\|_\infty}{\|\boldsymbol{x}\|_\infty},$$

$\boldsymbol{Ax} = \boldsymbol{b}$ 的**误差放大因子**(error magnification factor) 是两者的比率, 或

$$\text{误差放大因子} = \frac{\text{相对前向误差}}{\text{相对后向误差}} = \frac{\frac{\|\boldsymbol{x}-\boldsymbol{x}_c\|_\infty}{\|\boldsymbol{x}\|_\infty}}{\frac{\|\boldsymbol{r}\|_\infty}{\|\boldsymbol{b}\|_\infty}}. \tag{2.19}$$

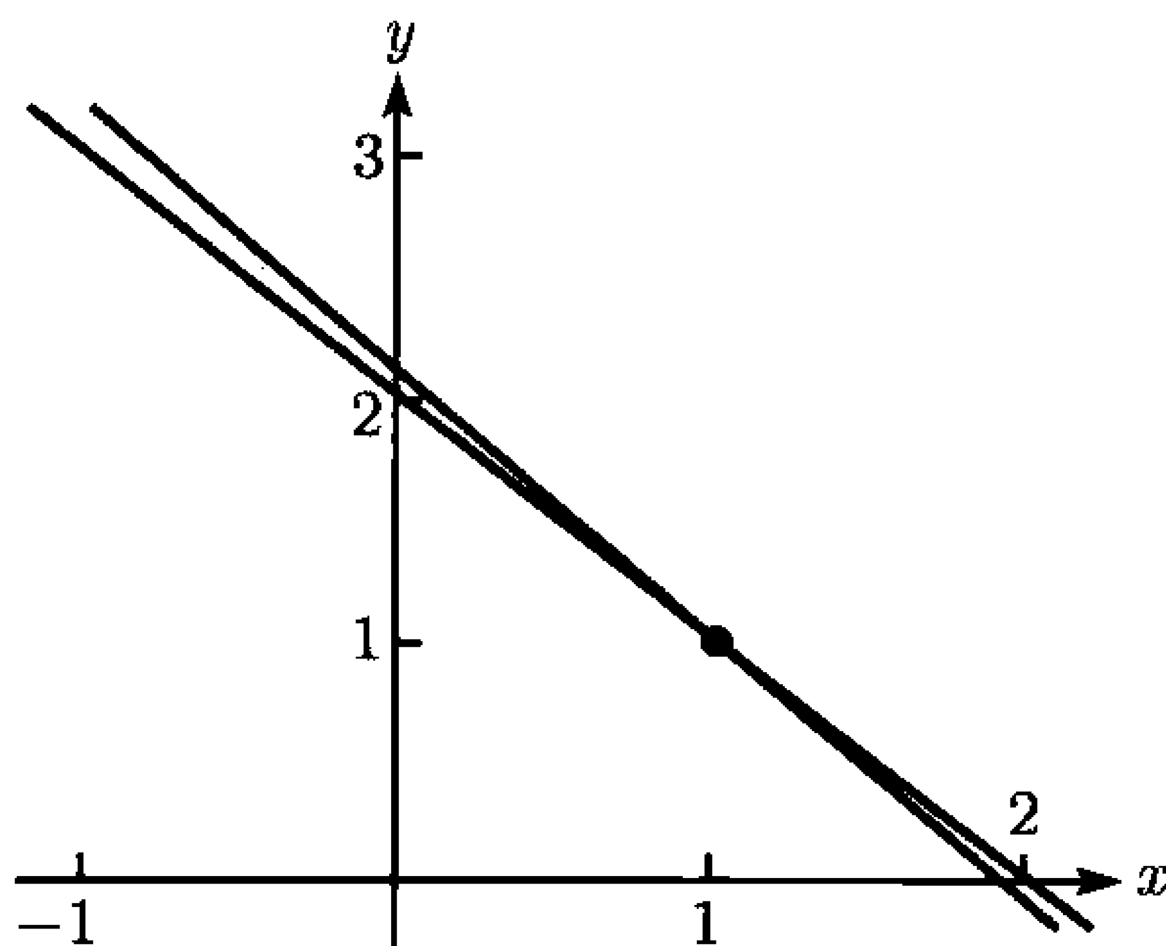


图 2-2 例 2.11 的几何性质. 直线 $x_2=2-x_1$ 与 $x_2=2.000\,1-1.000\,1x_1$ 代表方程组 (2.18), 它们交于 (1,1) 点. 点 $(-1, 3.000\ 1)$ 是一个解, 它几乎没有落在两条直线上. 图中两直线间的差异被夸大了——实际上它们非常接近

亮点 条件作用

条件数是贯穿数值分析始终的一个主题. 在第 1 章的关于 Wilkinson 多项式的讨论中, 对一个方程 $f(x) = 0$ 所给的小的扰动, 我们看到了如何在求根过程中计算误差放大因子. 对矩阵方程 $\boldsymbol{Ax} = \boldsymbol{b}$ 而言, 有类似的误差放大因子, 其最大可能的因子由 $\text{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ 给出.

对于方程组 (2.18), 相对后向误差为

$$\frac{0.000\ 1}{2.000\ 1} \approx 0.000\ 05 = 0.005\%,$$

相对前向误差为

$$\frac{2.000\ 1}{1} = 2.000\ 1 \approx 200\%,$$

误差放大因子为 $2.000\ 1/(0.000\ 1/2.000\ 1) = 40\ 004.000\ 1$.

在第 1 章中, 我们把条件数定义为在输入误差的规定范围内的最大误差放大率. "规定范围" 依赖于上下文. 现在在线性方程组的情形下, 我们将更精确地给出定义. 对一个固定的矩阵 $\boldsymbol{A}$, 考虑对不同的向量 $\boldsymbol{b}$ 求解 $\boldsymbol{Ax} = \boldsymbol{b}$. 在这种情况下, $\boldsymbol{b}$ 是输入而解 $\boldsymbol{x}$ 是输出. 输入中的一个小变化就是 $\boldsymbol{b}$ 的一个小变化, 它有一个误差放大因子. 因此我们得到下列定义.

定义 2.5 一个方阵 $\boldsymbol{A}$ 的**条件数** $\text{cond}(\boldsymbol{A})$ 是对所有的右端项 $\boldsymbol{b}$ 求解 $\boldsymbol{Ax}=\boldsymbol{b}$ 的最大可能误差放大因子.

令人惊讶的是, 对于方阵的条件数, 有一个简洁的公式. 类似于向量的范数, 定义一个 $n \times n$ 矩阵 $\boldsymbol{A}$ 的**矩阵范数**(matrix norm) 为

$$\|\boldsymbol{A}\|_\infty = \text{最大绝对值行和} \tag{2.20}$$

即对每一行把它的元素的绝对值全部加起来, 将这 n 个数的最大值指定为 $\boldsymbol{A}$ 的范数.

定理 2.6 $n \times n$ 矩阵 $\boldsymbol{A}$ 的条件数是

$$\text{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\|.$$

定理 2.6 将在后面予以证明, 使用它可以计算例 2.11 中的系数矩阵的条件数. 根据 (2.20),

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1.000\,1 & 1 \end{bmatrix}$$

的范数是 $\|\boldsymbol{A}\| = 2.000\,1$, $\boldsymbol{A}$ 的逆是

$$\boldsymbol{A}^{-1} = \begin{bmatrix} -10\,000 & 10\,000 \\ 10\,001 & -10\,000 \end{bmatrix},$$

其范数 $\|\boldsymbol{A}\|^{-1} = 20\,001$, $\boldsymbol{A}$ 的条件数为

$$\text{cond}(\boldsymbol{A}) = (2.000\,1)(20\,001) = 40\,004.000\,1.$$

这正是我们在例 2.11 中求出的误差放大率, 按条件数的定义, 它显然取到了最坏的情况. 这个方程组对任何其他的 $\boldsymbol{b}$ 的误差放大因子都小于或等于 40 004.000 1. 习题 3 要求计算其他一些误差放大因子.

条件数的有效数字与在第 1 章的是一样的. 具有数量 $\text{cond}(\boldsymbol{A})$ 大小的误差放大因子是可能的. 在浮点运算中, 不要期望相对后向误差比 $\varepsilon_{\text{mach}}$ 还要小, 因为 $\boldsymbol{b}$ 元素的存储就已经导致那么大的误差了. 根据 (2.19), 在求解 $\boldsymbol{Ax}=\boldsymbol{b}$ 时可能会产生大小为 $\varepsilon_{\text{mach}} \cdot \text{cond}(\boldsymbol{A})$ 的相对前向误差. 换而言之, 若 $\text{cond}(\boldsymbol{A}) \approx 10^k$, 那么在计算 x 时, 我们将损失 k 位精度.

在例 2.11 中, $\text{cond}(\boldsymbol{A}) \approx 4 \times 10^4$, 故按双精度, 我们将会得到 $16-4=12$ 位正确数字的解 $\boldsymbol{x}$. 可以引入 MATLAB 最好的有多种用途的线性方程组求解工具来验证这一点.

MATLAB 中的反斜线命令`x=A\b`利用我们将在 2.4 节中讨论的 $\boldsymbol{LU}$ 分解的一个先进的格式来求解线性方程组. 现在, 我们把它作为在浮点运算中我们可以得到的

最好可能的算法的一个例子. 下面的 MATLAB 命令给出了例 2.11 的计算机解 $\boldsymbol{x}_c$:

```
>> A = [1 1;1.0001 1]; b=[2;2.0001];
>> xc = A\b
xc =
   1.00000000000222
   0.99999999999778
```

与正确的解 $\boldsymbol{x}=[1,1]$ 相比, 计算机解有大约 11 位正确数字, 接近于由条件数推测出的结果.

Hilbert 矩阵 $\boldsymbol{H}$, 其元素 $H_{ij}=1/(i+j-1)$, 以大的条件数而著称.

例 2.12 设 $\boldsymbol{H}$ 为 $n\times n$ 的 Hilbert 矩阵. 利用 MATLAB 的反斜线命令计算 $\boldsymbol{Hx}=\boldsymbol{b}$ 的解, 其中 $\boldsymbol{b}=\boldsymbol{H}\cdot[1,\cdots,1]^{\mathrm{T}}, n=6$ 和 10.

右端项 $\boldsymbol{b}$ 的选择使得正确解为 n 个分量皆为 1 的向量, 这方便于检验前向误差. MATLAB 求出条件数 (按无穷大范数) 并计算出解:

```
>> n=6;H=hilb(n);
>> cond(H,inf)
ans =
    2.907027900294064e+007
>> b=H*ones(n,1);
>> xc=H\b
xc =
   0.99999999999923
   1.00000000002184
   0.99999999985267
   1.00000000038240
   0.99999999957855
   1.00000000016588
```

条件数大约为 10^7, 按最坏的情况推测有 $16-7=9$ 位正确数字. 计算出的解中确实有大约 9 位正确数字. 现按 $n=10$ 重复计算:

```
>> n=10;H=hilb(n);
>> cond(H,inf)
ans =
    3.535371683074594e+013
>> b=H*ones(n,1);
>> xc=H\b
xc =
   0.99999999875463
   1.00000010746631
   0.99999771299818
   1.00002077769598
   0.99990094548472
   1.00027218303745
   0.99955359665722
```

```
1.00043125589482
0.99977366058043
1.00004976229297
```

由于条件数为 10^{13}, 所以在解中仅有 $16-13=3$ 位正确数字.

对于比 10 稍大的 n, Hilbert 矩阵的条件数要比 10^{16} 更大, 因此不能保证在计算出的解中有正确数字. ◀

即使是出色的软件对病态问题也会束手无策. 增加精度会好点. 在扩展的精度下, $\varepsilon_{\text{mach}}=2^{-64}\approx 5.42\times 10^{-20}$, 我们以 20 而不是 16 位数字出发. 尽管如此, 由于 Hilbert 矩阵的条件数随 n 的增大而迅速地增大, 以至于最后任何适当的有限精度都将消失.

幸运的是, 像 Hilbert 矩阵那么大的条件数的情形是不多见的. 对 $n=10^4$ 及更大的含 n 个未知量的 n 个方程的良态的线性系统, 可以按双精度常规计算. 不过, 知道病态问题的存在性是重要的, 同时亦应了解条件数在判断其存在性的用处. 参见计算机问题 1~4 以获得更多关于误差放大率及条件数的例子.

本节所用到的无穷大向量范数是度量向量长度的一种简单方法, 它是**向量范数**(vector norm) 的一个例子, 向量范数 $\|\boldsymbol{x}\|$ 须满足下列 3 条性质:

(i) $\|\boldsymbol{x}\|\geqslant 0$, 当且仅当 $\boldsymbol{x}=[0,\cdots,0]$ 时 $\|\boldsymbol{x}\|=0$;

(ii) 对每一数量 α 及向量 $\boldsymbol{x}$, $\|\alpha\boldsymbol{x}\|=|\alpha|\cdot\|\boldsymbol{x}\|$;

(iii) 对向量 $\boldsymbol{x},\boldsymbol{y}$, $\|\boldsymbol{x}+\boldsymbol{y}\|\leqslant\|\boldsymbol{x}\|+\|\boldsymbol{y}\|$.

另外, $\|\boldsymbol{A}\|_\infty$ 是**矩阵范数**的一个例子, 矩阵范数要满足类似的 3 条性质:

(i) $\|\boldsymbol{A}\|\geqslant 0$, 当且仅当 $\boldsymbol{A}=\boldsymbol{0}$ 时, $\|\boldsymbol{A}\|=0$;

(ii) 对每一数量 α 及矩阵 $\boldsymbol{A}$, $\|\alpha\boldsymbol{A}\|=|\alpha|\cdot\|\boldsymbol{A}\|$;

(iii) 对矩阵 $\boldsymbol{A},\boldsymbol{B}$, $\|\boldsymbol{A}+\boldsymbol{B}\|\leqslant\|\boldsymbol{A}\|+\|\boldsymbol{B}\|$.

作为不同的例子, 向量 $\boldsymbol{x}=[x_1,\cdots,x_n]$ 的**向量 1- 范数**是 $\|\boldsymbol{x}\|_1=|x_1|+\cdots+|x_n|$. $n\times n$ 矩阵 $\boldsymbol{A}$ 的矩阵 1- 范数是 $\|\boldsymbol{A}\|_1=$ 最大绝对值列和, 即列向量的 1- 范数的最大值. 参见习题 9 和 10 证明这些条件定义了范数.

刚讨论过的误差放大因子、条件数和矩阵范数可对任意的向量和矩阵范数进行定义. 我们将对矩阵范数的讨论限于**算子范数**(operator norm), 即根据一种特殊的向量范数定义为

$$\|\boldsymbol{A}\|=\max\frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|},$$

其中最大值是在所有的非零向量 $\boldsymbol{x}$ 上取到. 则由定义, 矩阵范数与相应的向量范数是相容的, 意即

$$\|\boldsymbol{A}\boldsymbol{x}\|\leqslant\|\boldsymbol{A}\|\cdot\|\boldsymbol{x}\| \tag{2.21}$$

对任意的矩阵 $\boldsymbol{A}$ 及向量 $\boldsymbol{x}$ 成立. 习题 10 和习题 11 证明了由 (2.20) 定义的范数 $\|\boldsymbol{A}\|_\infty$ 不仅是一矩阵范数, 而且对无穷向量范数也是算子范数.

这个事实使得我们可以证明前面提到的 cond($\boldsymbol{A}$) 的简单表示. 下面的证明对无穷范数及其他任意的算子范数都成立.

定理 2.6 的证明 利用等式 $\boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}_c)=\boldsymbol{r}$ 及 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$, 由相容性 (2.21), 得

$$\|\boldsymbol{x}-\boldsymbol{x}_c\| \leqslant \|\boldsymbol{A}^{-1}\|\cdot\|\boldsymbol{r}\|,$$

$$\frac{1}{\|\boldsymbol{b}\|} \geqslant \frac{1}{\|\boldsymbol{A}\|\|\boldsymbol{x}\|}.$$

两个不等式置于一起得到

$$\frac{\|\boldsymbol{x}-\boldsymbol{x}_c\|}{\|\boldsymbol{x}\|} \leqslant \frac{\|\boldsymbol{A}\|}{\|\boldsymbol{b}\|}\|\boldsymbol{A}^{-1}\|\cdot\|\boldsymbol{r}\|,$$

这表明 $\|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ 是所有误差放大因子的一个上界. 第二点, 可以证明这个上界总是能取到的. 由算子范数的定义知, 可以选择 $\boldsymbol{x}$ 使得 $\|\boldsymbol{A}\|=\|\boldsymbol{A}\boldsymbol{x}\|/\|\boldsymbol{x}\|$ 及 $\boldsymbol{r}$ 使得 $\|\boldsymbol{A}^{-1}\|=\|\boldsymbol{A}^{-1}\boldsymbol{r}\|/\|\boldsymbol{r}\|$. 设 $\boldsymbol{x}_c=\boldsymbol{x}-\boldsymbol{A}^{-1}\boldsymbol{r}$, 则 $\boldsymbol{x}-\boldsymbol{x}_c=\boldsymbol{A}^{-1}\boldsymbol{r}$, 对 $\boldsymbol{x}$ 及 $\boldsymbol{r}$ 的这种特殊选取再检验下面的等式即可证明定理 2.6:

$$\frac{\|\boldsymbol{x}-\boldsymbol{x}_c\|}{\|\boldsymbol{x}\|}=\frac{\|\boldsymbol{A}^{-1}\boldsymbol{r}\|}{\|\boldsymbol{x}\|}=\frac{\|\boldsymbol{A}^{-1}\|\|\boldsymbol{r}\|\|\boldsymbol{A}\|}{\|\boldsymbol{A}\boldsymbol{x}\|}.$$

2.3.2 摆动

经典高斯消去法误差的第二个重要来源更易于确定. 我们用下面的例子来说明摆动.

例 2.13 考虑方程组

$$\begin{cases} 10^{-20}x_1+x_2=1, \\ x_1+2x_2=4. \end{cases}$$

我们将 3 次求解这个方程组: 一次用完全精度, 第二次根据 IEEE 双精度运算模拟计算机进行求解, 还有一次是我们首先交换方程的次序.

1. 精确解 高斯消元过程表形式为

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array}\right] \xrightarrow[10^{20}\times\text{第1行}]{\text{第2行减去}} \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2-10^{-20} & 4-10^{20} \end{array}\right],$$

底下的方程为

$$(2-10^{20})x_2=4-10^{20} \rightarrow x_2=\frac{4-10^{20}}{2-10^{20}},$$

上面的方程得到

$$10^{-20}x_1+\frac{4-10^{20}}{2-10^{20}}=1 \rightarrow x_1=10^{20}\left(1-\frac{4-10^{20}}{2-10^{20}}\right) \rightarrow x_1=\frac{-2\times10^{20}}{2-10^{20}}.$$

精确解为

$$[x_1, x_2] = \left[\frac{2 \times 10^{20}}{10^{20} - 2}, \frac{4 - 10^{20}}{2 - 10^{20}}\right] \approx [2, 1].$$

2. IEEE 双精度　高斯消元过程的计算机格式稍微有些不同:

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array}\right] \rightarrow \begin{array}{c} \text{第 2 行减去} \\ 10^{20} \times \text{第 1 行} \end{array} \rightarrow \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2 - 10^{20} & 4 - 10^{20} \end{array}\right].$$

按 IEEE 双精度, 由于舍入误差, $2 - 10^{20}$ 与 -10^{20} 是一样的. 类似地, $4 - 10^{20}$ 按 -10^{20} 被储存. 现在底下的方程是

$$-10^{20} x_2 = -10^{20} \rightarrow x_2 = 1.$$

上面方程的机器运算格式变成了

$$10^{-20} x_1 + 1 = 1,$$

所以 $x_1 = 0$, 计算出的解恰好为 $[x_1, x_2] = [0, 1]$.

与精确解相比, 这个解有较大的相对误差.

3. IEEE 双精度, 经行交换　经过交换两个方程的次序后, 我们重复高斯消元过程的计算机格式:

$$\left[\begin{array}{cc|c} 1 & 2 & 4 \\ 10^{-20} & 1 & 1 \end{array}\right] \rightarrow \begin{array}{c} \text{第 2 行减去} \\ 10^{-20} \times \text{第 1 行} \end{array} \rightarrow \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & 1 - 2 \times 10^{-20} & 1 - 4 \times 10^{-20} \end{array}\right]$$

按 IEEE 双精度, $1 - 2 \times 10^{-20}$ 被储存为 1, $1 - 4 \times 10^{-20}$ 也被储存为 1. 方程组现在是

$$\begin{cases} x_1 + 2x_2 = 4, \\ \quad\quad\; x_2 = 1, \end{cases}$$

这导出计算解 $x_1 = 2$ 和 $x_2 = 1$. 当然这不是精确答案. 但它直到大约 16 位数还是正确的, 这正是我们使用 52 位浮点数进行计算所能得到的最好结果.

最后两种计算之间的差异很大. 方法 3 给出了一个可接受的解而方法 2 没有. 方法 2 究竟出了什么样的差错? 进一步的分析引导我们把目光转向在消元步骤中用到的乘子 10^{20}. 底下的方程减去 10^{20} 乘以上面的方程的效果相当于去掉或 “淹没” 了底下的方程. 原先有两个独立的方程或信息的来源, 而经过方法 2 中的消元步骤后, 本质上得到的是上面那个方程的两种表示. 由于底下的方程已经消失, 故对于任何实用的目的, 我们不能期望计算解能满足底下的方程, 而计算解确实也不满足.

另一方面, 方法 3 完成消元过程而不摆动, 因为乘子是 10^{-20}. 经过消元后, 原来的两个方程仍然线性独立地存在, 只是稍作改变成了三角形. 其结果是一个更精确的近似解. ◀

例 2.13 说明了, 高斯消元中的乘子须尽可能地小, 以避免摆动. 幸好对原始的高斯消去法有一种简单的修正方法可使乘子的绝对值不超过 1. 这个在表形式中包含有审慎的行交换的新方法叫做部分主元法, 这是 2.4 节的主题.

习题 2.3

1. 求出下列每一个矩阵的范数 $\|\boldsymbol{A}\|_\infty$:

(a) $\boldsymbol{A}=\begin{bmatrix}1&2\\3&4\end{bmatrix}$; (b) $\boldsymbol{A}=\begin{bmatrix}1&5&1\\-1&2&-3\\1&-7&0\end{bmatrix}$.

2. 求出下列矩阵 (无穷范数) 的条件数.

(a) $\boldsymbol{A}=\begin{bmatrix}1&2\\3&4\end{bmatrix}$; (b) $\boldsymbol{A}=\begin{bmatrix}1&2.01\\3&6\end{bmatrix}$; (c) $\boldsymbol{A}=\begin{bmatrix}6&3\\4&2\end{bmatrix}$.

3. 就例 2.11 中方程组的下列近似解 $\boldsymbol{x}_c$(按无穷范数), 求出前向误差和后向误差以及误差放大因子:

(a) $[-1,3]$; (b) $[0, 2]$; (c) $[2, 2]$; (d) $[-2,4]$; (e) $[-2,4.000\,1]$.

4. 就方程组 $x_1+2x_2=1, 2x_1+4.01x_2=2$ 的下列近似解, 求出前向误差和后向误差以及误差放大因子:

(a) $[-1,1]$; (b) $[3,-1]$; (c) $[2,-\frac{1}{2}]$.

5. 就方程组 $x_1-2x_2=3, 3x_1-4x_2=7$ 的下列近似解, 求出相对前向误差和后向误差以及误差放大因子:

(a) $[-2,-4]$; (b) $[-2,-3]$; (c) $[0,-2]$; (d) $[-1,-1]$;

(e) 系统矩阵的条件数是什么?

6. 就方程组 $x_1+2x_2=3, 2x_1+4.01x_2=6.01$ 的下列近似解, 求出相对前向误差和相对后向误差以及误差放大因子:

(a) $[-10,6]$; (b) $[-100,52]$; (c) $[-600,301]$; (d) $[-599,301]$;

(e) 系数矩阵的条件数是什么?

7. 求出 5×5 的 Hilbert 矩阵的范数 $\|\boldsymbol{A}\|_\infty$.

8. (a) 求出方程组 $\begin{bmatrix}1&1\\1+\delta&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}2\\2+\delta\end{bmatrix}$ 的系数矩阵的条件数(作为 δ 的函数); (b) 求出近似根 $\boldsymbol{x}_c=[-1,3+\delta]$ 的误差放大因子.

9. (a) 证明无穷范数 $\|\boldsymbol{x}\|_\infty$ 是一向量范数; (b) 证明 1- 范数 $\|\boldsymbol{x}\|_1$ 是一向量范数.

10. (a) 证明无穷范数 $\|\boldsymbol{A}\|_\infty$ 是一无穷范数; (b) 证明 1- 范数 $\|\boldsymbol{A}\|_1$ 是一矩阵范数.

11. 证明矩阵无穷范数是向量无穷范数的算子范数.

12. 证明矩阵 1- 范数是向量 1- 范数的算子范数.

13. 对习题 1 中的矩阵, 求出满足 $\|\boldsymbol{A}\|_\infty=\|\boldsymbol{A}\boldsymbol{x}\|_\infty/\|\boldsymbol{x}\|_\infty$ 的一个向量 $\boldsymbol{x}$.

14. 对习题 1 中的矩阵, 求出满足 $\|\boldsymbol{A}\|_1=\|\boldsymbol{Ax}\|_1/\|\boldsymbol{x}\|_1$ 的一个向量 $\boldsymbol{x}$.

15. 求出

$$\boldsymbol{A}=\begin{bmatrix}10 & 20 & 1\\ 1 & 1.99 & 6\\ 0 & 50 & 1\end{bmatrix}$$

的 $\boldsymbol{LU}$ 分解, 所需要的最大的数量乘子 l_{ij} 是什么?

计算机问题 2.3

1. 对元素为 $A_{ij}=5/(i+2j-1)$ 的 $n\times n$ 矩阵, 设 $\boldsymbol{x}=[1,\cdots,1]^{\mathrm{T}}$ 及 $\boldsymbol{b}=\boldsymbol{Ax}$. 用计算机问题 2.1.1 中的 MATLAB 程序或 MATLAB 的反斜线命令, 计算双精度解 $\boldsymbol{x}_c$. 求出问题 $\boldsymbol{Ax}=\boldsymbol{b}$ 的前向误差和误差放大因子的无穷范数, 并与 $\boldsymbol{A}$ 的条件数进行比较. (a) $n=6$; (b) $n=10$.
2. 对元素 $A_{ij}=1/(|i-j|+1)$ 的矩阵, 执行计算机问题 2.1.1.
3. 设 $\boldsymbol{A}$ 是元素为 $A_{ij}=|i-j|+1$ 的 $n\times n$ 矩阵, 定义 $\boldsymbol{x}=[1,\cdots,1]^{\mathrm{T}}$ 及 $\boldsymbol{b}=\boldsymbol{Ax}$. 对 $n=100,200,300,400,500$, 用计算机问题 2.1.1 中的 MATLAB 程序或 MATLAB 的反斜线命令, 计算双精度解 $\boldsymbol{x}_c$. 对每个解算出前向误差的无穷范数. 求出问题 $\boldsymbol{Ax}=\boldsymbol{b}$ 的 5 个误差放大因子, 并与相应的条件数进行比较.
4. 对元素为 $A_{ij}=\sqrt{(i-j)^2+n/10}$ 的矩阵, 执行计算机问题 3 的步骤.
5. 对多大的 n, 计算机问题 1 中的解没有正确的有效数字?
6. 用计算机问题 2.1.1 中的 MATLAB 程序, 按双精度执行并完成例 2.13 中方法 2 与方法 3, 并与正文中求出的理论结果进行比较.

2.4 *PA=LU* 分解

迄今考虑的高斯消去法的形式经常被称为是 “原始” 的, 这是由于有两个须认真对待的困难: 碰到一个零主元及出现摆动问题. 对一非奇异矩阵而言, 这两者皆可由一改进的算法而避免. 这种改进的关键是一种对系数矩阵作行交换的有效技术, 叫做部分选主元 (法).

2.4.1 部分选主元

在对含 n 个未知量的 n 个方程运用经典的高斯消去法的一开始, 第一步是用对角元 a_{11} 作为主元消去第一列. **部分选主元**(partial pivoting) 方法在执行每一步消元步骤之前要比较数的大小, 确定第一列中最大元素所在的位置, 并将其所在的行交换到主元行 (这里即是第一行).

换言之, 在高斯消去法的开始, 部分主元法要求我们选择第 p 行, 其中

$$|a_{p1}|\geqslant|a_{i1}| \tag{2.22}$$

(对所有的 $1 \leqslant i \leqslant n$), 并交换第 1 行和第 p 行. 接着, 像通常做的, 用新的 a_{11} 作为主元进行第 1 列的消元过程. 用于消去 a_{i1} 的乘子将是

$$m_{i1} = \frac{a_{i1}}{a_{11}}$$

且

$$|m_{i1}| \leqslant 1.$$

在算法进程中, 每次主元的选择都要作同样的检验. 当决定第二次选主元时, 我们从当前的 a_{22} 开始并检查它下面的所有元素, 我们选择第 p 行使得

$$|a_{p2}| \geqslant |a_{i2}|$$

(对所有的 $2 \leqslant i \leqslant n$), 若 $p \neq 2$, 则交换第 2 行与第 p 行. 这一步骤中不再涉及第 1 行. 若 $|a_{22}|$ 已是最大元素, 则不必进行行交换.

在消元过程中, 这种技术应用于每一列. 在消去第 k 列之前, 对 $k \leqslant p \leqslant m$ 的 p, 确定最大的 $|a_{pk}|$ 所在的位置, 如果需要, 在继续消元之前交换第 p 行和第 k 行. 注意到使用部分主元法保证了所有的乘子或 $\boldsymbol{L}$ 的元素在绝对值的意义下都将不大于 1. 借助于实现高斯消元的这种小小的改变, 将完全避免例 2.13 中说明的摆动问题.

例 2.14 应用带有部分选主元的高斯消去法求解方程组 (2.1).

方程组可写成表形式为

$$\left[\begin{array}{rr|r} 1 & 1 & 3 \\ 3 & -4 & 2 \end{array}\right],$$

根据部分选主元法, 我们比较 $|a_{11}| = 1$ 与它下面所有元素的大小, 而在该例中仅有一个元素 $a_{21} = 3$. 因 $|a_{21}| > |a_{11}|$, 我们必须交换第 1 行和第 2 行. 新的表为

$$\left[\begin{array}{rr|r} 3 & -4 & 2 \\ 1 & 1 & 3 \end{array}\right] \longrightarrow \begin{array}{c} \text{第2行减去} \\ \frac{1}{3} \times \text{第1行} \end{array} \longrightarrow \left[\begin{array}{rr|r} 3 & -4 & 2 \\ 0 & \frac{7}{3} & \frac{7}{3} \end{array}\right].$$

经回代, 解为 $x_2 = 1$ 以及 $x_1 = 2$, 这正是先前已求出的. 在我们第一次求解这个方程组时, 乘子为 3, 而在部分选主元下, 这将不再出现. ◀

例 2.15 应用带有部分选主元的高斯消去法求解方程组

$$\begin{cases} x_1 - x_2 + 3x_3 = -3, \\ -x_1 - 3x_3 = 1, \\ 2x_1 + 2x_2 + 4x_3 = 0. \end{cases}$$

本例写成表形式为

$$\left[\begin{array}{rrr|r} 1 & -1 & 3 & -3 \\ -1 & 0 & -2 & 1 \\ 2 & 2 & 4 & 0 \end{array}\right].$$

由部分主元法, 我们比较 $|a_{11}| = 1$ 与 $|a_{21}| = 1$ 及 $|a_{31}| = 2$, 选择 a_{31} 为新的主元, 这通过第 1 行和第 3 行的交换而实现:

$$\left[\begin{array}{rrr|r} 1 & -1 & 3 & -3 \\ -1 & 0 & -2 & 1 \\ 2 & 2 & 4 & 0 \end{array}\right] \rightarrow \text{交换第 1 行和第 3 行} \rightarrow \left[\begin{array}{rrr|r} 2 & 2 & 4 & 0 \\ -1 & 0 & -2 & 1 \\ 1 & -1 & 3 & -3 \end{array}\right]$$

$$\rightarrow \begin{array}{c}\text{第 2 行减去} \\ (-\frac{1}{2})\times\text{第 1 行}\end{array} \rightarrow \left[\begin{array}{rrr|r} 2 & 2 & 4 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & -1 & 3 & -3 \end{array}\right] \rightarrow \begin{array}{c}\text{第 3 行减去} \\ \frac{1}{2}\times\text{第 1 行}\end{array} \rightarrow \left[\begin{array}{rrr|r} 2 & 2 & 4 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & -2 & 1 & -3 \end{array}\right].$$

在消去第 2 列之前, 我们必须比较当前的 $|a_{22}|$ 与 $|a_{32}|$, 由于后者较大, 我们又一次交换行:

$$\left[\begin{array}{rrr|r} 2 & 2 & 4 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & -2 & 1 & -3 \end{array}\right] \rightarrow \begin{array}{c}\text{交换第 2 行} \\ \text{和第 3 行}\end{array} \rightarrow \left[\begin{array}{rrr|r} 2 & 2 & 4 & 0 \\ 0 & -2 & 1 & -3 \\ 0 & 1 & 0 & 1 \end{array}\right]$$

$$\rightarrow \begin{array}{c}\text{第 3 行减去} \\ (-\frac{1}{2})\times\text{第 2 行}\end{array} \rightarrow \left[\begin{array}{rrr|r} 2 & 2 & 4 & 0 \\ 0 & -2 & 1 & -3 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{array}\right].$$

注意到所有 3 个乘子的绝对值都小于 1.

方程组现在易于求解了. 由

$$\begin{cases} \frac{1}{2}x_3 = -\frac{1}{2}, \\ -2x_2 + x_3 = -3, \\ 2x_1 + 2x_2 + 4x_3 = 0, \end{cases}$$

我们解出 $x = [1, 1, -1]$. ◀

注意到部分主元法也解决了零主元的问题. 当碰到零主元时, 例如, 若 $a_{11} = 0$, 则立即与它所在列的某处的非零主元进行行交换. 若在对角线处或其下方没有这样的非零元素, 则该矩阵是奇异的, 高斯消去法将不会给出一个解.

2.4.2 置换矩阵

在说明行交换如何在用于高斯消去法的 LU 分解方法之前, 我们将讨论置换矩阵的基本性质.

定义 2.7 一个置换矩阵(permutation matrix) 是除了在每一行每一列只有一个 1 外其余全为 0 的 $n \times n$ 的矩阵.

等价地, 一个置换矩阵 $\boldsymbol{P}$ 可由对 $n \times n$ 的单位矩阵施以任意的行交换 (或任意的列交换) 而生成. 例如,

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

是仅有的 2×2 的置换矩阵, 而

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

是 6 个 3×3 的置换矩阵.

下一个定理告诉我们, 在一个矩阵的左边乘上一个置换矩阵的作用是什么.

定理 2.8 **置换矩阵的基本定理** 设 $\boldsymbol{P}$ 是由对单位矩阵作一系列特定的行交换而形成的 $n \times n$ 的置换矩阵, 则对任意的 $n \times n$ 矩阵 $\boldsymbol{A}$, $\boldsymbol{PA}$ 表示把完全相同的一系列行交换作用于 $\boldsymbol{A}$ 而得到的矩阵.

例如, 置换矩阵

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

是单位矩阵经交换 2, 3 两行后形成的. 在任一矩阵的左边乘上 $\boldsymbol{P}$ 相当于交换 2, 3 两行:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ g & h & i \\ d & e & f \end{bmatrix}.$$

记住定理 2.8 的一种好方法是想象 $\boldsymbol{P}$ 乘上单位矩阵:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

看待这个等式有两种不同的方法: 首先, 看作乘以单位矩阵 (故我们在右边得到置换矩阵); 其次, 看作是作用在单位矩阵的行上面的置换矩阵. 定理 2.8 的内容是: 乘以 $\boldsymbol{P}$ 所引起的行交换恰好就是形成 $\boldsymbol{P}$ 时所进行的那些行交换.

2.4.3　$PA = LU$ 分解

本节把对高斯消去法所知的一切融合起来得到 $PA = LU$ 分解, 这是部分主元消去法的矩阵公式. $PA = LU$ 分解是求解线性方程组的重要方法.

顾名思义, $PA = LU$ 分解就是 A 的行交换方法的 LU 分解. 在用部分主元方法时, 一开始我们并不知道需要交换的行, 所以必须很细心地使行交换的信息与分解保持一致. 特别地, 当进行一次行交换时, 我们需要记下前面的乘子. 我们从一个例子开始.

例 2.16　求出矩阵

$$A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}$$

的 $PA = LU$ 分解.

首先, 1, 2 两行需要交换, 根据部分主元法:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & -1 \end{bmatrix} \to \text{交换第 1 行和第 2 行} \to \begin{bmatrix} 4 & 4 & -4 \\ 2 & 1 & 5 \\ 1 & 3 & 1 \end{bmatrix}.$$

我们将用置换矩阵 P 记录这样下去已经累计进行的行交换. 下面演示两次行运算, 即

$$\to \begin{array}{c}\text{第 2 行减去} \\ \frac{1}{2} \times \text{第 1 行}\end{array} \to \begin{bmatrix} 4 & 4 & -4 \\ \textcircled{\tfrac{1}{2}} & -1 & 7 \\ 1 & 3 & 1 \end{bmatrix} \to \begin{array}{c}\text{第 3 行减去} \\ \frac{1}{4} \times \text{第 1 行}\end{array} \to \begin{bmatrix} 4 & 4 & -4 \\ \textcircled{\tfrac{1}{2}} & -1 & 7 \\ \textcircled{\tfrac{1}{4}} & 2 & 2 \end{bmatrix}$$

以消去第 1 列. 我们已经有了一些新的做法——并不是仅仅把零放在消去的位置上, 而是把零作为一个存储地点. 在 (i, j) 位置的零里面, 我们存储了用于消元的乘子 m_{ij}. 我们这样做是有理由的. 通过这种方法可以将乘子置于它们所在的行, 以进行未来的行交换.

接着我们必须作一比较以选出第二个主元. 由于 $|a_{22}| = 1 < 2 = |a_{32}|$, 在消去第二列前需进行一次行交换.

注意到前面的乘子随着行交换而移动:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \to \text{交换第 2 行和第 3 行} \to \begin{bmatrix} 4 & 4 & -4 \\ \textcircled{\tfrac{1}{4}} & 2 & 2 \\ \textcircled{\tfrac{1}{2}} & -1 & 7 \end{bmatrix}$$

$$\to \text{第 3 行减去} \left(-\frac{1}{2}\right) \times \text{第 2 行} \to \begin{bmatrix} 4 & 4 & -4 \\ \textcircled{\frac{1}{4}} & 2 & 2 \\ \textcircled{\frac{1}{2}} & \textcircled{-\frac{1}{2}} & 8 \end{bmatrix}.$$

这就完成了消元. 现在可很快给出 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解

$$\underset{\boldsymbol{P}}{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}} \underset{\boldsymbol{A}}{\begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}} = \underset{\boldsymbol{L}}{\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}} \underset{\boldsymbol{U}}{\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix}}. \tag{2.23}$$

$\boldsymbol{L}$ 上的元素置于矩阵的下三角部分的零里面 (在主对角线下), 而 $\boldsymbol{U}$ 来自上三角. 最终 (累积) 的置换矩阵记作 $\boldsymbol{P}$. ◀

利用 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解求解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 只是 $\boldsymbol{A}=\boldsymbol{LU}$ 方法的一个微小的变形. 对方程 $\boldsymbol{Ax}=\boldsymbol{b}$ 左乘 $\boldsymbol{P}$, 然后如前进行:

$$\begin{aligned} \boldsymbol{PAx} &= \boldsymbol{Pb}, \\ \boldsymbol{LUx} &= \boldsymbol{Pb}, \end{aligned} \tag{2.24}$$

解 (1) $\boldsymbol{Lc}=\boldsymbol{Pb}$ 得到 $\boldsymbol{c}$; (2) $\boldsymbol{Ux}=\boldsymbol{c}$ 得到 $\boldsymbol{x}$. (2.25)

正如以前提及的, 重要的是计算中花费最多的部分——决定 $\boldsymbol{PA}=\boldsymbol{LU}$ 可以在不知道 $\boldsymbol{b}$ 的情况下完成. 由于结果得到的是 $\boldsymbol{PA}$(即方程组系数矩阵的行置换) 的 $\boldsymbol{LU}$ 分解, 所以在进行回代过程之前, 对右端向量 $\boldsymbol{b}$ 按完全相同的方式进行置换是必要的, 这可在回代步骤的第一步利用 $\boldsymbol{Pb}$ 来实现. 高斯消去法的矩阵公式的价值是显然的: 消元和选主元过程中所需要记录下的细节均自动包含在矩阵方程中.

例 2.17 利用 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解求解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$, 其中

$$\boldsymbol{A} = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix}.$$

由 (2.23) 可知 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解. 还需要完成两次回代.

(1) $\boldsymbol{Lc}=\boldsymbol{Pb}$:

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 5 \end{bmatrix}.$$

从上面的方程开始, 我们有

$$\begin{cases} c_1 = 0, \\ \frac{1}{4}(0) + c_2 = 6 \Rightarrow c_2 = 6, \\ \frac{1}{2}(0) - \frac{1}{2}(6) + c_3 = 5 \Rightarrow c_3 = 8. \end{cases}$$

(2) $\boldsymbol{Ux} = \boldsymbol{c}$:

$$\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 8 \end{bmatrix}.$$

从底下的方程开始,

$$\begin{cases} 8x_3 = 8 \Rightarrow x_3 = 1, \\ 2x_2 + 2(1) = 6 \Rightarrow x_2 = 2, \\ 4x_1 + 4(2) - 4(1) = 0 \Rightarrow x_1 = -1. \end{cases} \tag{2.26}$$

因此, 解为 $\boldsymbol{x} = [-1, 2, 1]$. ◀

例 2.18 利用带有部分选主元的 $\boldsymbol{PA} = \boldsymbol{LU}$ 分解求解方程组 $2x_1 + 3x_2 = 4, 3x_1 + 2x_2 = 1$.

按矩阵形式, 这是方程

$$\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}.$$

开始时我们先不管右端项 $\boldsymbol{b}$. 根据部分选主元法, 1, 2 两行必须进行交换 (因为 $a_{21} > a_{11}$). 消元步骤为

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \boldsymbol{A} = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix} \rightarrow \text{交换 } 1, 2 \text{ 两行} \rightarrow \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$$

$$\rightarrow \begin{array}{c} \text{第 2 行减去} \\ \frac{2}{3}\times\text{第 1行} \end{array} \rightarrow \begin{bmatrix} 3 & 2 \\ \textcircled{\frac{2}{3}} & \frac{5}{3} \end{bmatrix}.$$

因此, $\boldsymbol{PA} = \boldsymbol{LU}$ 分解为

$$\underset{\boldsymbol{P}}{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}} \underset{\boldsymbol{A}}{\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}} = \underset{\boldsymbol{L}}{\begin{bmatrix} 1 & 0 \\ \frac{2}{3} & 1 \end{bmatrix}} \underset{\boldsymbol{U}}{\begin{bmatrix} 3 & 2 \\ 0 & \frac{5}{3} \end{bmatrix}}$$

第一次回代 $Lc = Pb$ 为

$$\begin{bmatrix} 1 & 0 \\ \frac{2}{3} & 1 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

从上面开始, 我们有

$$\begin{cases} c_1 = 1, \\ \frac{2}{3}(1) + c_2 = 4 \Rightarrow c_2 = \frac{10}{3}. \end{cases}$$

第二次回代 $Ux = c$ 为

$$\begin{bmatrix} 3 & 2 \\ 0 & \frac{5}{3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{10}{3} \end{bmatrix}.$$

从下面开始, 我们有

$$\begin{cases} \frac{5}{3}x_2 = \frac{10}{3} \Rightarrow x_2 = 2, \\ 3x_1 + 2(2) = 1 \Rightarrow x_1 = -1. \end{cases} \tag{2.27}$$

因此, 解为 $x = [-1, 2]$. ◀

每个 $n \times n$ 矩阵都有 $PA = LU$ 分解. 由部分选主元规则可知, 若选出的主元为零, 这意味着所需要消去的元素都已为零, 则该列就已处理完毕.

迄今为止所有叙述的技术都可以在 MATLAB 中实现. 我们已讨论过的高斯消去法的最复杂的形式是 $PA = LU$ 分解. MATLAB 的lu命令接受一个方形系数矩阵 A 并输出 P, L 及 U. 下面的 MATLAB 程序段定义了例 2.16 的矩阵, 并计算了它的分解.

```
>> A=[2 1 5; 4 4 -4; 1 3 1];
>> [L,U,P]=lu(A)

L=

    1.0000         0         0
    0.2500    1.0000         0
    0.5000   -0.5000    1.0000

U=

     4     4    -4
     0     2     2
     0     0     8
P=

     0     1     0
     0     0     1
     1     0     0
```

习题 2.4

1. 求出下列矩阵的 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解 (利用部分主元法):

(a) $\begin{bmatrix}1&3\\2&3\end{bmatrix}$; (b) $\begin{bmatrix}2&4\\1&3\end{bmatrix}$; (c) $\begin{bmatrix}1&5\\5&12\end{bmatrix}$; (d) $\begin{bmatrix}0&1\\1&0\end{bmatrix}$.

2. 求出下列矩阵的 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解 (利用部分主元法):

(a) $\begin{bmatrix}1&1&0\\2&1&-1\\-1&1&-1\end{bmatrix}$; (b) $\begin{bmatrix}0&1&3\\2&1&1\\-1&-1&2\end{bmatrix}$;

(c) $\begin{bmatrix}1&2&-3\\2&4&2\\-1&0&3\end{bmatrix}$; (d) $\begin{bmatrix}0&1&0\\1&0&2\\-2&1&0\end{bmatrix}$.

3. 由求出的 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解, 求解方程组并执行两步回代过程:

(a) $\begin{bmatrix}3&7\\6&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}1\\-11\end{bmatrix}$; (b) $\begin{bmatrix}3&1&2\\6&3&4\\3&1&5\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}0\\1\\3\end{bmatrix}$.

4. 由求出的 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解, 求解方程组并执行两步回代过程:

(a) $\begin{bmatrix}4&2&0\\4&4&2\\2&2&3\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}2\\4\\6\end{bmatrix}$; (b) $\begin{bmatrix}-1&0&1\\2&1&1\\-1&2&0\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}-2\\17\\3\end{bmatrix}$.

5. 写下一个 5×5 的矩阵 $\boldsymbol{P}$, 使其左乘另一矩阵的作用为进行 2, 5 两行的交换.

6. (a) 写下一个 4×4 的矩阵 $\boldsymbol{P}$, 使其左乘一个矩阵的作用为那个矩阵的第 2 行与第 4 行进行交换; (b) 用 $\boldsymbol{P}$ 进行右乘的效果是什么? 举例说明.

7. 改变最左边矩阵的 4 个元素, 使矩阵方程正确:

$$\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\begin{bmatrix}1&2&3&4\\3&4&5&6\\5&6&7&8\\7&8&8&0\end{bmatrix}=\begin{bmatrix}5&6&7&8\\3&4&5&6\\7&8&9&0\\1&2&3&4\end{bmatrix}.$$

8. 求出习题 2.3.15 中矩阵 $\boldsymbol{A}$ 的 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解, 所需的最大乘子 l_{ii} 是什么?

9. (a) 求出 $\boldsymbol{A}=\begin{bmatrix}1&0&0&1\\-1&1&0&1\\-1&-1&1&1\\-1&-1&-1&1\end{bmatrix}$ 的 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解; (b) 设 $\boldsymbol{A}$ 为形如 (a) 中形式的 $n\times n$ 矩阵, 描述其 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解中每个矩阵的元素.

10. (a) 假设 $\boldsymbol{A}$ 为一个 $n\times n$ 矩阵, 其元素 $|a_{ij}|\leqslant 1$, 对于 $1\leqslant i,j\leqslant n$. 证明其 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解中的矩阵 $\boldsymbol{U}$ 满足 $|u_{ij}|\leqslant 2^{n-1}$, 对所有的 $1\leqslant i,j\leqslant n$. 参见习题 9(b). (b) 对任一 $n\times n$ 矩阵 $\boldsymbol{A}$ 用公式表示并证明一个类似的结论.

2.5 迭代方法

高斯消去法是一个 $O(n^3)$ 的浮点运算的有限序列, 并最终给出一个解. 因此, 高斯消去法叫做求解线性方程组的直接方法. 理论上直接方法在有限步之内给出精确解 (当然, 当在一有限精度的计算机上执行时, 得到的结果将只是近似的. 这正如我们前面已见, 精度的损失由条件数度量). 直接方法与第 1 章所描述的形式上为迭代的求根方法形成了对照.

所谓的迭代方法也可用于求解线性方程组. 类似于不动点迭代, 迭代方法从某一初始估计出发, 每一步对其进行改善直至收敛到解向量.

2.5.1 Jacobi 方法

Jacobi 方法是求解方程组的不动点迭代的一种形式. 不动点迭代 (FPI) 的第一步是改写方程组, 求解未知量. Jacobi 方法的第一步按下列标准方式进行: 求解第 i 个方程以得到第 i 个未知量; 然后如不动点迭代一样, 从某一初始估计开始进行迭代.

例 2.19 应用 Jacobi 方法解方程组 $3u+v=5, u+2v=5$

通过求解第一个方程得到 u, 求解第二个方程得到 v. 使用初始估计 $(u_0, v_0) = (0,0)$, 我们有

$$\begin{cases} u = \frac{5-v}{3}, \\ v = \frac{5-u}{2}. \end{cases} \tag{2.28}$$

迭代两个方程:

$$\begin{aligned}
\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\
\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}, \\
\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/2}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix}, \\
\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} &= \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-5/6}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix}.
\end{aligned} \tag{2.29}$$

继续 Jacobi 迭代过程表明它收敛到解 [1, 2]. ◀

现在假设方程组按相反的次序给出.

例 2.20 应用 Jacobi 方法解方程组 $u+2v=5, 3u+v=5$.

解第一个方程得到第一个变量 u, 解第二个方程得到 v. 我们从

$$\begin{cases} u = 5 - 2v, \\ v = 5 - 3u \end{cases} \tag{2.30}$$

开始, 如前一样迭代两个方程, 但结果却大不相同:

$$\begin{aligned}
\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\
\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} &= \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \\
\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} &= \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}, \\
\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} &= \begin{bmatrix} 5 - 2(-10) \\ 5 - 3(-5) \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}.
\end{aligned} \tag{2.31}$$

在这种情况下 Jacobi 方法失败了, 因为迭代发散了. ◀

由于 Jacobi 方法并不总能成功, 所以有必要了解在什么条件下它能成功. 其中一个重要的条件由下列定义给出:

定义 2.9 若对每个 $1 \leqslant i \leqslant n, |a_{ii}| > \sum\limits_{j \neq i} |a_{ij}|$, 则称 $n \times n$ 矩阵 $\boldsymbol{A} = (a_{ij})$ 是**严格对角占优**的. 也就是说, 每个主对角元在绝对值上要比所在行的其他所有元素的绝对值和更大.

定理 2.10 若 $n \times n$ 矩阵 $\boldsymbol{A}$ 是严格对角占优的, 则 (1) $\boldsymbol{A}$ 为非奇异矩阵; (2) 对每个向量 $\boldsymbol{b}$ 及每个初始估计, 应用于 $\boldsymbol{Ax} = \boldsymbol{b}$ 上的 Jacobi 方法收敛到 (唯一的) 解.

定理 2.10 说明, 若 $\boldsymbol{A}$ 是严格对角占优的, 则应用于方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 上的 Jacobi 方法对每个初始估计均收敛到一个解. 这个事实的证明在 2.5.3 节给出. 在例 2.19 中, 系数矩阵首先是

$$\boldsymbol{A} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix},$$

因为 $3 > 1, 2 > 1$, 故它是严格对角占优的. 在这种情况下, 收敛性得到了保证. 另一方面, 在例 2.20 中, 将 Jacobi 方法用于矩阵

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix},$$

这个矩阵不是对角占优的, 不存在这样 (收敛性) 的保证. 注意到严格对角占优只是一个充分条件, 当没有这个条件时, Jacobi 方法仍可收敛.

例 2.21 确定矩阵

$$A=\begin{bmatrix}3&1&-1\\2&-5&2\\1&6&8\end{bmatrix},\qquad B=\begin{bmatrix}3&2&6\\1&8&1\\9&2&-2\end{bmatrix}$$

是否严格对角占优.

矩阵 $\boldsymbol{A}$ 是严格对角占优的, 因为 $|3|>|1|+|-1|, |-5|>|2|+|2|, |8|>|1|+|6|$. $\boldsymbol{B}$ 不是的, 因为, 例如说 $|3|>|2|+|6|$ 是不对的. 可是, 如果交换 $\boldsymbol{B}$ 的第一行和第三行, 则 $\boldsymbol{B}$ 就是对角占优的, 且可保证 Jacobi 方法收敛. ◀

Jacobi 方法是不动点迭代的一种形式. 设 $\boldsymbol{D}$ 表示 $\boldsymbol{A}$ 的主对角部分, $\boldsymbol{L}$ 表示 $\boldsymbol{A}$ 的下三角部分 (主对角线下方的元素), $\boldsymbol{U}$ 表示上三角部分 (主对角线上方的部分), 则 $\boldsymbol{A}=\boldsymbol{L}+\boldsymbol{D}+\boldsymbol{U}$, 要解的方程为 $\boldsymbol{Lx}+\boldsymbol{Dx}+\boldsymbol{Ux}=\boldsymbol{b}$. 注意这里 $\boldsymbol{L}$ 与 $\boldsymbol{U}$ 的用法与 $\boldsymbol{LU}$ 分解中的用法是不同的, 因为这里 $\boldsymbol{L}$ 与 $\boldsymbol{U}$ 的对角线元素都为零. 方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 可改写成不动点迭代的形式:

$$\begin{aligned}\boldsymbol{Ax}&=\boldsymbol{b},\\(\boldsymbol{D}+\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}&=\boldsymbol{b},\\\boldsymbol{Dx}&=\boldsymbol{b}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x},\\\boldsymbol{x}&=\boldsymbol{D}^{-1}(\boldsymbol{b}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}).\end{aligned}\tag{2.32}$$

由于 $\boldsymbol{D}$ 是对角矩阵, 故其逆为由 $\boldsymbol{A}$ 的对角元的倒数所成的矩阵. Jacobi 方法只是 (2.32) 的不动点迭代.

Jacobi 方法

$$\begin{aligned}\boldsymbol{x}_0&=\text{初始向量},\\\boldsymbol{x}_{k+1}&=\boldsymbol{D}^{-1}(\boldsymbol{b}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}_k),\quad k=0,1,2,\cdots.\end{aligned}\tag{2.33}$$

对于例 2.19,

$$\begin{bmatrix}3&1\\1&2\end{bmatrix}\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}5\\5\end{bmatrix},$$

带有 $\boldsymbol{x}_k=\begin{bmatrix}u_k\\v_k\end{bmatrix}$ 的不动点迭代 (2.33) 为

$$\begin{bmatrix}u_{k+1}\\v_{k+1}\end{bmatrix}=\boldsymbol{D}^{-1}(\boldsymbol{b}-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}_k)=\begin{bmatrix}\frac{1}{3}&0\\0&\frac{1}{2}\end{bmatrix}\left(\begin{bmatrix}5\\5\end{bmatrix}-\begin{bmatrix}0&1\\1&0\end{bmatrix}\begin{bmatrix}u\\v\end{bmatrix}_k\right)=\begin{bmatrix}\frac{5-v_k}{3}\\\frac{5-u_k}{2}\end{bmatrix},$$

这与我们原先的格式是一致的.

2.5.2 Gauss-Seidel 方法和 SOR

与 Jacobi 方法密切相关的一种迭代叫做 Gauss-Seidel 方法. Gauss-Seidel 方法与 Jacobi 方法之间仅有的差别是, 前者在每一步用到最新校正过的未知量的值, 即便是校正发生在当前步. 回到例 2.19, 我们看到 Gauss-Seidel 方法像这样:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{3} \end{bmatrix},$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-10/9}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{35}{18} \end{bmatrix},$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-v_2}{3} \\ \frac{5-u_3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-35/18}{3} \\ \frac{5-55/54}{2} \end{bmatrix} = \begin{bmatrix} \frac{55}{54} \\ \frac{215}{108} \end{bmatrix}. \tag{2.34}$$

注意 Gauu-Seidel 与 Jacobi 之间的差别: v_1 的定义用到 u_1 而不是 u_0. 我们看到了用 Jacobi 方法得到解 [1, 2] 的方法, 但这儿相同的步数会有稍微更高的精度. 若 Gauss-Seidel 方法是收敛的, 它经常比 Jacobi 收敛得更快. 定理 2.11 证明, 只要系数矩阵是严格对角占优的, Gauss-Seidel 方法将和 Jacobi 一样收敛到解.

Gauss-Seidel 可以写成矩阵形式, 并被认为是一种不动点迭代, 其中我们分离方程 $(\boldsymbol{L}+\boldsymbol{D}+\boldsymbol{U})\boldsymbol{x}=\boldsymbol{b}$ 为

$$(\boldsymbol{L}+\boldsymbol{D})\boldsymbol{x}_{k+1} = -\boldsymbol{U}\boldsymbol{x}_k + \boldsymbol{b}.$$

注意到在用到刚刚新确定的元素 $\boldsymbol{x}_{k+1}$ 的同时, 也置 $\boldsymbol{A}$ 的下三角部分于左端了. 重新安排方程给出了 Gauss-Seidel 方法.

Gauss-Seidel 方法

$$\boldsymbol{x}_0 = \text{初始向量},$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{D}^{-1}(\boldsymbol{b} - \boldsymbol{U}\boldsymbol{x}_k - \boldsymbol{L}\boldsymbol{x}_{k+1}), \quad k = 0, 1, 2, \cdots.$$

例 2.22 应用 Gauss-Seidel 方法, 求解方程组

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}.$$

Gauss-Seidel 迭代为

$$u_{k+1}=\frac{4-v_k+w_k}{3},\qquad v_{k+1}=\frac{1-2u_{k+1}-w_k}{4},\qquad w_{k+1}=\frac{1+u_{k+1}-2v_{k+1}}{5}.$$

从 $\boldsymbol{x}_0=[u_0,v_0,w_0]=[0,0,0]$ 开始, 我们计算

$$\begin{bmatrix}u_1\\v_1\\w_1\end{bmatrix}=\begin{bmatrix}\frac{4-0-0}{3}=\frac{4}{3}\\\frac{1-8/3-0}{4}=-\frac{5}{12}\\\frac{1+4/3+5/6}{5}=\frac{19}{30}\end{bmatrix}\approx\begin{bmatrix}1.3333\\-0.4167\\0.6333\end{bmatrix},$$

$$\begin{bmatrix}u_2\\v_2\\w_2\end{bmatrix}=\begin{bmatrix}\frac{101}{60}\\-\frac{3}{4}\\\frac{251}{300}\end{bmatrix}\approx\begin{bmatrix}1.6833\\-0.7500\\0.8367\end{bmatrix}.$$

方程组是严格对角占优的, 因此迭代将收敛到解 $[2,-1,1]$. ◀

称为**逐次超松弛**(Successive Over-Relaxation, SOR) 的方法采取 Gauss-Seidel 趋向于解的方向并试图加速收敛. 设 ω 是一实数, 定义新估计量 $\boldsymbol{x}_{k+1}$ 的每个分量为 ω 乘上 Gauss-Seidel 公式与 $1-\omega$ 乘上当前估计量 $\boldsymbol{x}_k$ 的加权平均. 数 ω 叫做**松弛参数**(relaxation parameter), $\omega>1$ 时被认为是**超松弛的**(over-relaxation).

例 2.23 取 $\omega=1.25$, 应用 SOR 求解例 2.22 的方程组.

由逐次超松弛法得到

$$\begin{aligned}u_{k+1}&=(1-\omega)u_k+\omega\frac{4-v_k+w_k}{3},\\v_{k+1}&=(1-\omega)v_k+\omega\frac{1-2u_{k+1}-w_k}{4},\\w_{k+1}&=(1-\omega)w_k+\omega\frac{1+u_{k+1}-2v_{k+1}}{5}.\end{aligned}$$

从 $[u_0,v_0,w_0]=[0,0,0]$ 开始, 我们计算得到

$$\begin{bmatrix}u_1\\v_1\\w_1\end{bmatrix}\approx\begin{bmatrix}1.6667\\-0.7292\\1.0312\end{bmatrix},\qquad\begin{bmatrix}u_2\\v_2\\w_2\end{bmatrix}\approx\begin{bmatrix}1.9835\\-1.0672\\1.0216\end{bmatrix}.$$

在本例中, SOR 迭代比 Jacobi 和 Gauss-Seidel 更快地收敛到解 $[2,-1,1]$. ◀

正如 Jacobi 和 Gauss-Seidel 一样, SOR 的另一推导来自把方程组当作一不动点问题. 问题 $\boldsymbol{Ax}=\boldsymbol{b}$ 可写成 $(\boldsymbol{L}+\boldsymbol{D}+\boldsymbol{U})\boldsymbol{x}=\boldsymbol{b}$, 用 ω 相乘并重排,

$$\begin{aligned}(\omega\boldsymbol{L}+\omega\boldsymbol{D}+\omega\boldsymbol{U})\boldsymbol{x}&=\omega\boldsymbol{b},\\(\omega\boldsymbol{L}+\boldsymbol{D})\boldsymbol{x}&=\omega\boldsymbol{b}-\omega\boldsymbol{U}\boldsymbol{x}+(1-\omega)\boldsymbol{D}\boldsymbol{x},\\\boldsymbol{x}&=(\omega\boldsymbol{L}+\boldsymbol{D})^{-1}[(1-\omega)\boldsymbol{D}\boldsymbol{x}-\omega\boldsymbol{U}\boldsymbol{x}]+\omega(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}\boldsymbol{b}.\end{aligned}$$

逐次超松弛 (SOR)

$x_0 =$ 初始向量,

$x_{k+1} = (\omega L + D)^{-1}[(1-\omega)Px_k - \omega(x_k)] + \omega(D+\omega L)^{-1}b, \quad k = 0, 1, 2, \cdots.$

$\omega = 1$ 时的 SOR 恰好是 Gauss-Seidel. 在叫做逐次低松弛(Successive Under-Relaxation) 的方法中可允许参数 ω 小于 1.

例 2.24　对含 6 个未知量的 6 个方程的方程组, 比较 Jacobi、Gauss-Seidel 和 SOR:

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}. \tag{2.35}$$

解是 $x = [1,1,1,1,1,1]$. 这 3 种方法各自运行 6 步后得到的近似解向量 x_6 如表 2-1 所示.

表　2-1

Jacobi	Gauss-Seidel	SOR
0.987 9	0.995 0	0.998 9
0.984 6	0.994 6	0.999 3
0.967 4	0.996 9	1.000 4
0.967 4	0.999 6	1.000 9
0.984 6	1.001 6	1.000 9
0.987 9	1.001 3	1.000 4

逐次超松弛方法中的参数 w 设为 1.1. 对这个问题, SOR 显示出了它的优越性. ◀

图 2-3 就不同的 ω, 比较了经过 6 步迭代后例 2.24 的无穷范数误差. 尽管没有

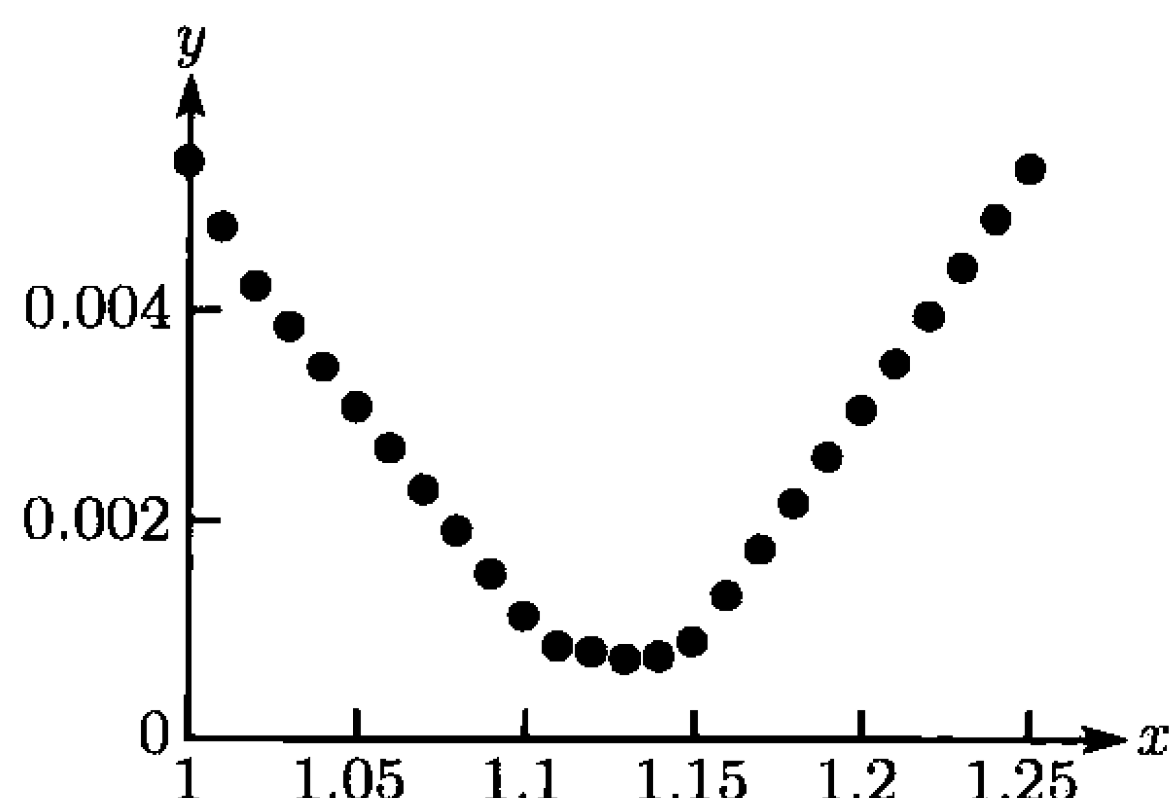


图 2-3　对例 2.24, 经过 6 步 SOR 迭代后的无穷范数误差, 它是超松弛参数 ω 的一个函数. Gauss-Seidel 对应于 $\omega = 1$, 最小误差发生在 $\omega \approx 1.13$ 时

一般性理论来描述 ω 的最优选择, 但在这里显然有最优选择. 对一些常用的特殊情况下的最优 ω 的讨论, 参见 [10].

2.5.3 迭代方法的收敛性

本节证明: 对于严格对角占优矩阵, Jacobi 和 Gauss-Seidel 方法是收敛的. 这就是定理 2.10 与定理 2.11 的内容.

Jacobi 方法写作

$$\boldsymbol{x}_{k+1} = -\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}_k + \boldsymbol{D}^{-1}\boldsymbol{b}. \tag{2.36}$$

附录 A 的定理 A.7 决定了这样一种迭代的收敛性. 根据这个定理, 我们需要知道谱半径

$$\rho(\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})) < 1$$

以保证 Jacobi 方法的收敛性. 而严格对角占优矩阵恰好蕴含了这一点, 接下来我们会加以证明.

定理 2.10 的证明 设 $\boldsymbol{R} = \boldsymbol{L}+\boldsymbol{U}$ 表示矩阵的非对角部分. 为检验 $\rho(\boldsymbol{D}^{-1}\boldsymbol{R}) < 1$, 设 λ 为 $\boldsymbol{D}^{-1}\boldsymbol{R}$ 的一个特征值, $\boldsymbol{v}$ 为相应的特征向量. 选择这个 $\boldsymbol{v}$ 使得 $\|\boldsymbol{v}\|_\infty = 1$, 并使得对一些 $1 \leqslant m \leqslant n$, 分量 $v_m = 1$ 而所有其他的分量都不大于 1(由任一特征向量开始, 并除以最大的分量即可做到这一点. 一特征向量的任意常数倍还是同一特征值的特征向量). 特征值的定义意味着 $\boldsymbol{D}^{-1}\boldsymbol{R}\boldsymbol{v} = \lambda\boldsymbol{v}$ 或 $\boldsymbol{R}\boldsymbol{v} = \lambda\boldsymbol{D}\boldsymbol{v}$.

因 $r_{mm} = 0$, 取这个向量方程的第 m 个分量的绝对值意味着:

$$\begin{aligned}&|r_{m1}v_1 + r_{m2}v_2 + \cdots + r_{m,m-1}v_{m-1} + r_{m,m+1}v_{m+1} + \cdots + r_{mn}v_n|\\ =\ &|\lambda d_{mm}v_m| = |\lambda||d_{mm}|.\end{aligned}$$

由于所有的 $|v_i| \leqslant 1$, 左端项至多为 $\sum\limits_{j\neq m}|r_{mj}|$, 而根据严格对角占优的假设, 它小于 $|d_{mm}|$, 这蕴含着 $|\lambda||d_{mm}| < |d_{mm}|$, 这又使得 $|\lambda| < 1$. 因为 λ 是任一特征值, 我们已如愿证明了 $\rho(\boldsymbol{D}^{-1}\boldsymbol{R}) < 1$. 现在附录 A 的定理 A.7 蕴含着 Jacobi 收敛到 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ 的解. 最后, 因为 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ 对任一 $\boldsymbol{b}$ 都有一个解, 故 $\boldsymbol{A}$ 为非奇异矩阵.

把 Gauss-Seidel 方法置入到 (2.36) 的形式中, 得到

$$\boldsymbol{x}_{k+1} = -(\boldsymbol{L}+\boldsymbol{D})^{-1}\boldsymbol{U}\boldsymbol{x}_k + (\boldsymbol{L}+\boldsymbol{D})^{-1}\boldsymbol{b}.$$

那么很清楚的是, 如果矩阵

$$(\boldsymbol{L}+\boldsymbol{D})^{-1}\boldsymbol{U} \tag{2.37}$$

的谱半径小于 1, 就得到了 Gauss-Seidel 方法的收敛性. 下一个定理说明严格对角占优性蕴含了对特征值的这个要求.

定理 2.11 若 $n \times n$ 矩阵 $\boldsymbol{A}$ 是严格对角占优的, 则 (1) $\boldsymbol{A}$ 是非奇异矩阵, (2) 对每个向量 $\boldsymbol{b}$ 及每个开始猜测, 应用于 $\boldsymbol{Ax}=\boldsymbol{b}$ 的 Gauss-Seidel 方法都收敛到一个解.

证 设 λ 为 (2.37) 的一个特征值, 相应的特征向量为 $\boldsymbol{v}$. 如前一定理中的证明那样, 选择特征向量使得 $v_m=1$ 且所有其他的分量在绝对值上都小于 1. 注意到 $\boldsymbol{L}$ 的元素是 $a_{ij}, i>j$; $\boldsymbol{U}$ 的元素是 $a_{ij}, i<j$. 则仔细检查 (2.37) 的特征值方程

$$\lambda(\boldsymbol{D}+\boldsymbol{L})\boldsymbol{v}=\boldsymbol{U}\boldsymbol{v}$$

的 m 行得到类似于前一证明中的一串不等式

$$\begin{aligned}|\lambda|\left(\sum_{i>m}|a_{mi}|\right) &< |\lambda|\left(|a_{mm}|-\sum_{i<m}|a_{mi}|\right) \leqslant |\lambda|\left(|a_{mm}|-\left|\sum_{i<m}a_{mi}v_i\right|\right)\\ &\leqslant |\lambda|\left|a_{mm}+\sum_{i<m}a_{mi}v_i\right| = \left|\sum_{i>m}a_{mi}v_i\right| \leqslant \sum_{i>m}|a_{mi}|.\end{aligned}$$

这就得到 $|\lambda|<1$, 从而完成了证明.

2.5.4 稀疏矩阵计算

基于高斯消去法的直接方法提供了在有限步内就可得到解的方法. 寻求迭代方法的理由是什么? 仅仅是为了得到近似解或者是收敛所需要的若干步骤?

使用像 Gauss-Seidel 这样的迭代方法有两个主要的理由. 这两个理由来自于这样的事实, 即迭代方法的每一步仅需要完整的 $\boldsymbol{LU}$ 分解方法所需浮点运算的一部分. 正如在本章前面所述, 求解 $n \times n$ 矩阵的高斯消去法花费 n^3 阶的运算. 而 Jacobi 方法的一步, 例如, 需大约 n^2 次乘法 (对每个矩阵元素) 及大约相同次数的加法. 问题是在用户的容忍范围内需要多少步才能得到收敛性.

对一种迭代方法特别值得关注的情况是何时已知得到了一个好的近似解. 例如, 在得到了 $\boldsymbol{Ax}=\boldsymbol{b}$ 的解之后, $\boldsymbol{A}$ 或 $\boldsymbol{b}$ 发生了一些小的变化. 我们可以想象一个动态问题, 其中 $\boldsymbol{A}$ 和 $\boldsymbol{b}$ 发生变化时不断地被重新度量, 且不断地得到一个精确的校正解 $\boldsymbol{x}$. 如果前一个问题的解被用作为类似的新问题的初始估计, 则可期望得到 Jacobi 或 Gauss-Seidel 方法更好的收敛性.

假设问题 (2.35) 中的 $\boldsymbol{b}$ 从最初的 $\boldsymbol{b}=[2.5,1.5,1,1,1.5,2.5]$ 经一小小的变化而变成了新的 $\boldsymbol{b}=[2.2,1.6,0.9,1.3,1.4,2.45]$. 我们可检验到该方程组的真解从 $[1,1,1,1,1,1]$ 变成了 $[0.9,1,1,1.1,1,1]$. 假设我们存储有以前一表中经 6 步 Gauss-Seidel 迭代后的解 $\boldsymbol{x}_6$, 把它用作初始估计, 对新的 $\boldsymbol{b}$ 继续 Gauss-Seidel 迭代, 并借助于有用的初始估计 $\boldsymbol{x}_6$ 仅需再迭代一步就可得到一个好的近似. 接下来的两步如表 2-2.

表 2-2

x_7	x_8
0.898 0	0.899 4
0.998 0	0.988 9
0.965 9	0.992 7
1.089 2	1.096 6
0.997 1	1.000 5
0.999 3	1.000 3

这种方法常常叫做**精化**(polishing), 因为该方法从一近似解 (它可能是前面相关问题的解) 开始, 仅对其加工改善使其更精确. 在实时应用场合, 即当数据随时间在更新而需要重复求解同样的问题时, 精化方法是常用的. 若系统规模大而时间短, 则在规定的时间内运行一个完整的高斯消去法或甚至一个回代过程都不可能. 如果解还没有太大的改变, 那么当解随时间变化时, 经过几步相对便宜的迭代方法后, 解可保持足够的精度.

求解稀疏方程组 (sparse systems of equations) 是使用迭代方法的第二个主要的理由. 如果一个系数矩阵的许多元素已知为 0, 则此矩阵叫做是**稀疏**的. 经常一个稀疏矩阵有 n^2 个元素为 0, 而仅有 $O(n)$ 个非零元素. 相反的情况是**满**的矩阵, 其少数的元素可能为 0. 对一稀疏矩阵应用高斯消去法通常会导致**填充**(fill-in), 即由于必要的行运算而使得系数矩阵从稀疏变为满的. 因此, 通常对稀疏矩阵避免使用高斯消去法及 $\boldsymbol{PA}=\boldsymbol{LU}$ 方法, 而把迭代方法作为可行的替代方法.

例 2.24 可扩展为下例中的稀疏矩阵.

例 2.25 用 Jacobi 方法求解例 2.24 的含有 100 000 个方程的形式.

设 n 为一偶数, 考虑 $n\times n$ 矩阵 $\boldsymbol{A}$, 其对角元为 3, 超对角元 (superdiagonal) 和次对角元 (subdiagonal) 为 -1, 对除 $i=\frac{n}{2}$ 及 $\frac{n}{2}+1$ 外所有的 $i=1,2,\cdots,n$, $(i,n+1-i)$ 位置上的元素为 $\frac{1}{2}$. 例如 $n=12$ 时,

$$\boldsymbol{A}=\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix}. \tag{2.38}$$

定义向量 $\boldsymbol{b}=(2.5,1.5,\cdots,1.5,1.0,1.0,1.5,\cdots,1.5,2.5)$, 其中有 $n-4$ 个 1.5 和 2 个 1.0. 注意到若 $n=6$, 则 $\boldsymbol{A}$ 和 $\boldsymbol{b}$ 定义了例 2.24 的方程组. 对一般的 n, 方程组的解为 $[1,\cdots,1]$.$\boldsymbol{A}$ 的行中非零元素没有多于 4 个的. 由于 n^2 个可能的元素中只有不超过 $4n$ 个元素是非零的, 我们可称矩阵 $\boldsymbol{A}$ 是稀疏的.

如果我们想对 $n=100\ 000$ 或更大的 n 求解这个方程组, 该选择什么方法呢? 若把系数矩阵 $\boldsymbol{A}$ 看作是满的则意味着要存 $n^2=10^{10}$ 个元素, 而每个元素作为一个浮点双精度数需要 8 字节的存储空间. 注意到 8×10^{10} 个字节近似于 80 个 G. 根据你的计算机配置, 也许不可能把所有的 n^2 个元素都存在 RAM(随机存储器) 中.

我们面临的困难不仅是矩阵太大, 还有时间. 高斯消去法所需的运算次数将是 $n^3\approx10^{15}$ 阶的. 如果你的机器以几 GHz(每秒 10^9 次) 运行, 则每秒浮点运算次数的一个上界大约为 10^8. 因此, $10^{15}/10^8=10^7$ 就是高斯消去法所需秒数的一个合理的估计. 一年也不过有 3×10^7 秒, 虽然这是一个粗略的计算, 但很明显, 对这个问题, 高斯消去法耗费的时间太长了!!

另一方面, 迭代方法的每一步大约需要 $2\times4n=800\ 000$ 次运算, 即对每个非零的矩阵元素进行两次运算. 我们可进行 Jacobi 迭代 100 步, 且还是在不到 10^8 次运算后即可完成, 这将在一台现代个人电脑上花费大约 1 秒不到的时间. 对刚才定义的 $n=100\ 000$ 的方程组, 下面的 Jacobi 代码`Jacobi.m`仅需 50 步即可从一初始估计 $(0,\cdots,0)$ 收敛到解 $(1,\cdots,1)$, 精确到小数点后 6 位, 在一般的个人电脑上执行 50 步所需要的时间不到 1 秒钟.

```
% Program 2.1 Sparse matrix setup
% Input: n = size of system
% Outputs: sparse matrix a, r.h.s. b
function [a,b] = sparsesetup(n)
e = ones(n,1);
n2=n/2;
a = spdiags([-e 3*e -e],-1:1,n,n);   % Entries of a
c=spdiags([e/2],0,n,n);c=fliplr(c);a=a+c;
a(n2+1,n2) = -1; a(n2,n2+1) = -1;    % Fix up 2 entries
b=zeros(n,1);                         % Entries of r.h.s. b
b(1)=2.5;b(n)=2.5;b(2:n-1)=1.5;b(n2:n2+1)=1;
```

```
% Program 2.2 Jacobi Method
% Inputs: full or sparse matrix a, r.h.s. b,
%         number of Jacobi iterations, k
% Output: solution x
function x = jacobi(a,b,k)
n=length(b);        % find n
d=diag(a);          % extract diagonal of a
```

```
r=a-diag(d);        % r is the remainder
x=zeros(n,1);       % initialize vector x
for j=1:k           % loop for Jacobi iteration
  x = (b-r*x)./d;
end                 % End of Jacobi iteration loop
```

注意一下上述代码的一些有意思的地方. 程序sparsesetup.m使用了 MATLAB 的 spdiags命令, 该命令把矩阵 $\boldsymbol{A}$ 定义为一稀疏数据结构. 本质上这意味着矩阵由一组三元数组 (i,j,d) 表示, 其中 d 为矩阵在位置 (i,j) 处的实数元素. 存储器并不提供给全部的 n^2 个可能的元素, 而只是基于所需进行配置. 命令spdiags将矩阵的列置于主对角线或其下方或上方的特定的次对角线或超对角线上.

MATLAB 的矩阵操作命令可无缝对接地用于稀疏矩阵数据结构. 例如, 用 MATLAB的lu命令对上述代码进行修改即可直接求解方程组. 不过对那个例子, 尽管 $\boldsymbol{A}$ 是稀疏的, 在程序执行过程中, 由高斯消去法得到的上三角矩阵 $\boldsymbol{U}$ 仍出现了填充现象. 例如, 对前面的矩阵 $\boldsymbol{A}$, 当 $n=12$ 时, 由高斯消去得到的上三角矩阵 $\boldsymbol{U}$ 为

$$
\begin{bmatrix}
3 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 \\
0 & 2.7 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0.165 \\
0 & 0 & 2.6 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0.187 & 0.062 \\
0 & 0 & 0 & 2.6 & -1.000 & 0 & 0 & 0 & 0.500 & 0.191 & 0.071 & 0.024 \\
0 & 0 & 0 & 0 & 2.618 & -1.000 & 0 & 0.500 & 0.191 & 0.073 & 0.027 & 0.009 \\
0 & 0 & 0 & 0 & 0 & 2.618 & -1.000 & 0.191 & 0.073 & 0.028 & 0.010 & 0.004 \\
0 & 0 & 0 & 0 & 0 & 0 & 2.618 & -0.927 & 0.028 & 0.011 & 0.004 & 0.001 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.562 & -1.032 & -0.012 & -0.005 & -0.001 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.473 & -1.047 & -0.018 & -0.006 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.445 & -1.049 & -0.016 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.440 & -1.044 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.458
\end{bmatrix}.
$$

由于 $\boldsymbol{U}$ 变成了一个相对较满的矩阵, 前面提到的存储约束又成了一个限制. 为完成求解过程, 需要用到 n^2 个存储位置的有效部分来存放 $\boldsymbol{U}$. 用迭代方法求解这个大型稀疏系统会更有效, 在执行时间及存储量上可相差几个数量级. ◀

习题 2.5

1. 用初始向量 $[0,\cdots,0]$ 计算 Jacobi 和 Gauss-Seidel 方法的前两步:

(a) $\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}=\begin{bmatrix} 5 \\ 4 \end{bmatrix}$; **(b)** $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \\ w \end{bmatrix}=\begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$;

(c) $\begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix}\begin{bmatrix} u \\ v \\ w \end{bmatrix}=\begin{bmatrix} 6 \\ 3 \\ 5 \end{bmatrix}$.

2. 重排方程组形成一个严格对角占优系统. 用初始向量 $[0,\cdots,0]$ 计算 Jacobi 和 Gauss-Seidel 方法的前两步:

(a) $\begin{cases} u+3v=-1, \\ 5u+4v=6; \end{cases}$　(b) $\begin{cases} u-8v-2w=1, \\ u+v+5w=4, \\ 3u-v+w=-2; \end{cases}$　(c) $\begin{cases} u+4v=5, \\ v+2w=2, \\ 4u+3w=0. \end{cases}$

3. 利用 SOR 求解习题 1 中的方程组, 计算前两步, 用初始向量 $[0,\cdots,0]$ 及 $\omega=1.5$.

4. 对重排后的习题 2 计算 SOR 的前两步, 用初始向量 $[0,\cdots,0]$ 及 $\omega=1$ 和 1.2.

5. 设 λ 为 $n\times n$ 矩阵 $\boldsymbol{A}$ 的一个特征值.

(a) 证明 Gershgorin 圆盘定理：存在一个对角元 A_{mm} 使得 $|A_{mm}-\lambda|\leqslant\sum\limits_{j\neq m}|A_{mj}|$ (提示：从一特征向量 $\boldsymbol{v}$ 开始, 使得 $\|\boldsymbol{v}\|_\infty=1$, 如定理 2.10 中的证明).

(b) 证明严格对角占优矩阵不会有零特征值. 这是定理 2.10 部分 (1) 的另一证明.

计算机问题 2.5

1. 用 Jacobi 方法解稀疏方程组, 精确到小数点 6 位 (前向误差按无穷范数), 其中分别取 $n=100$ 及 $n=100\ 000$. 正确的解是 $[1,\cdots,1]$. 给出所需的步数及后向误差. 方程组为

$$\begin{bmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 3 & -1 \\ & & & -1 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} 2 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ 2 \end{bmatrix}.$$

2. 用 Jacobi 方法求解稀疏方程组, 精确到小数点 3 位 (前向误差按无穷范数), 其中 $n=100$. 正确解为 $[1,-1,1,-1,\cdots,1,-1]$. 给出所需的步数及后向误差. 方程组为

$$\begin{bmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 2 & 1 \\ & & & 1 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ -1 \end{bmatrix}.$$

3. 改写程序 2.2 以执行 Gauss-Seidel 迭代. 求解例 2.24 中的问题来检验你的工作.

4. 改写程序 2.2 以执行 SOR. 取 $\omega=1.1$ 再检验例 2.24.

5. 用 (a) Gauss-Seidel 方法和 (b) $\omega=1.2$ 的 SOR, 对 $n=100$ 执行计算机问题 1 的步骤.

6. 用 (a) Gauss-Seidel 方法和 (b) $\omega=1.5$ 的 SOR, 执行计算机问题 2 的步骤.

7. 利用计算机问题 3 中你的程序, 确定用 Gauss-Seidel 方法在 1 秒钟的计算中你能精确求解多大规模的类似于 (2.28) 的方程组. 给出所需的时间及对不同 n 值的前向误差.

实例检验 2 Euler-Bernoulli 梁

Euler-Bernoulli 梁是在应力作用下弯曲的一个简单模型. 离散化把微分方程模型转化成了一个线性方程组. 减小离散化的步长则给出越来越大的方程组. 令人惊讶的是, 还没等到计算时间成为一个 (需要考虑的) 因素之前, 系统的规模已被坏条件限制住了.

梁的垂直位移由一个函数 $y(x)$ 表示, 其中 $0 \leqslant x \leqslant L$(沿着长为 L 的梁的方向). 我们将在计算中使用 MKS 单位 (米, 千克, 秒). 位移 $y(x)$ 满足 Euler-Bernoulli 方程

$$EIy'''' = f(x), \tag{2.39}$$

其中, 材料的杨氏模数 E 及惯性的面积矩 I 为沿着梁的常量. 右端项 $f(x)$ 为应用负荷, 包括梁的重量 (每单位长度上的力).

离散化导数的方法可以在第 5 章中找到, 在那里将证明 4 阶导数的一个合理近似为

$$y''''(x) \approx \frac{y(x-2h) - 4y(x-h) + 6y(x) - 4y(x+h) + y(x+2h)}{h^4} \tag{2.40}$$

对小的增量 h 成立 (见习题 5.1.19). 这种近似的离散化误差与 h^2 成比例. 我们的策略将是把梁看作为许多长度为 h 的小部分的并, 把微分方程的离散化形式用于每一个小部分.

对一正整数 n, 设 $h = \frac{L}{n+1}$. 考虑等分的网格 $0 = x_0 < x_1 < \cdots < x_n < x_{n+1} = L$, 其中 $h = x_i - x_{i-1}, i = 1, \cdots, n$. 对位移 $y_i = y(x_i)$ 用差分近似 (2.40) 代替微分方程 (2.39) 得到线性方程组为

$$y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2} = \frac{h^4}{EI} f(x_i) \tag{2.41}$$

其中 $i = 1, \cdots, n$. 有 n 个未知量 $y_1, \cdots, y_n$, 以及 n 个方程. 系数矩阵或结构矩阵的元素将来自这个方程的左端项, 稍作改变以考虑关于梁端点处支撑的假设.

首先考虑带有下列边界条件的梁

$$y(0) = y'(0) = y(L) = y'(L) = 0,$$

这叫做**固定梁**(pinned beam). 在两端它是固定的, 其斜率被约束为 0. Euler-Bernoulli 方程允许在中间的下垂度的计算.

固定梁的端点条件用近似

$$y''''(x) \approx \frac{12y(x+h) - 6y(x+2h) + \frac{4}{3}y(x+3h)}{h^4}$$

来处理, 其中 $x = 0$. 当 $y(x) = y'(x) = 0$ 时, 近似是有效的, 正如习题 5.1.20(a) 所证. 因此, 对 $i = 1$, 考虑到了端点条件的 (2.41) 形式为

$$12y_1 - 6y_2 + \frac{4}{3}y_3 = \frac{h^4}{EI} f(x_0).$$

对梁的右端作同样的考虑, 则对固定梁得到下列方程组:

$$
\begin{bmatrix}
12 & -6 & \frac{4}{3} & & & & & & \\
-4 & 6 & -4 & 1 & & & & & \\
1 & -4 & 6 & -4 & 1 & & & & \\
 & 1 & -4 & 6 & -4 & 1 & & & \\
 & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\
 & & & 1 & -4 & 6 & -4 & 1 & \\
 & & & & 1 & -4 & 6 & -4 & 1 \\
 & & & & & 1 & -4 & 6 & -4 \\
 & & & & & & \frac{4}{3} & -6 & -12
\end{bmatrix}
\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}
= \frac{h^4}{EI}
\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix}. \tag{2.42}
$$

结构矩阵 A 是一个**带状矩阵**(banded matrix), 即离开主对角线足够远处的所有元素皆为 0. 在 (2.42) 中, 除了 $|i-j| \leqslant 2$ 外, 矩阵元素 $a_{ij}=0$, 这个带状矩阵的**带宽**(bandwidth) 为 5.

现在讨论一个具体的例子, 考虑长 10 米、高 5 厘米、宽 10 厘米的立体钢梁. 钢的密度大约为 7 850 kg/m^3(1 牛顿力为 1kg $\cdot$ m/sec^2, 钢的杨氏模量大约为 2×10^{11} 帕斯卡或 N/m^2), 在其质心周围的惯性的面积矩 I 为 $bh^3/12$, 其中 d 是梁的高度, b 是宽度.

我们开始计算没有有效负载的梁的位移, 使 $f(x)$ 仅代表梁本身的重量. 幸运的是, 在那种情形下, (2.39) 有一个闭型解, 使得我们可以比较计算结果的精度. 每米梁的重量 f 是重力加速度 9.81m/sec^2 与常量 7 850bd kg/m 的乘积. 读者可以验证 (2.39) 两边的单位是一致的.

建议习题

1. 使用 MATLAB 的**spdiags**命令, 写一段 MATLAB 程序来定义 (2.42) 中的结构矩阵 A. 试用 Jacobi 和 Gauss-Seidel 方法, 求解 $n=10$ 的方程组以得到位移 y_i. 哪一个是收敛的? 收敛精确到小数点 6 位需要多少步?
2. 相对正确解 $y(x)=fx^2(L-x)^2/(24EI)$ 画出第 1 步的解, 其中 $f=f(x)$ 是前面定义的常数. 注意到梁的位移将是非负数, 所以下垂梁的实图将反转位移的符号. 计算梁的中点 $x=5$ 米处的误差.
3. 对 $n=10\times2^k$ 再运行步骤 1 中的计算, 其中 $k=1,\cdots,11$. 对每个 n, 画出在 $x=5$ 处的误差的表. 为什么误差随 n 而增大? 你可以作出 A 的条件数的一个伴随表以帮助回答最后一个问题.
4. 现在以 $x=6$ 到 $x=7$ 给梁加上 1 000 kg 的有效负载. 你必须对所有的 $6\leqslant x_i\leqslant 7$ 给 $f(x_i)$ 加上 1 000 kg/m$^2\times$9.81 m/sec^2, 并用步骤 3 中的 n 的值再次求解该问题. 绘出解. 沿着梁在何处发生最大位移?
5. 现在解开横梁的一端. 在右端带有自由边界条件的悬臂梁 (cantilever beam) 满足条件

$$y(0)=y'(0)=y''(L)=y'''(L)=0.$$

悬臂梁方程组 (关于求导见习题 5.1.20) 为

$$\begin{bmatrix} 12 & -6 & \frac{4}{3} & & & & & & \\ -4 & 6 & -4 & 1 & & & & & \\ 1 & -4 & 6 & -4 & 1 & & & & \\ & 1 & -4 & 6 & -4 & 1 & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & 1 & -4 & 6 & -4 & 1 & \\ & & & & 1 & -4 & 6 & -4 & 1 \\ & & & & & 1 & -\frac{93}{25} & \frac{111}{25} & -\frac{43}{25} \\ & & & & & & \frac{12}{25} & -\frac{24}{25} & \frac{12}{25} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \frac{h^4}{EI} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix} . \tag{2.43}$$

对解除负荷的悬臂梁重复步骤 3, 并与正确的解 $y(x) = (f/24EI)x^2(x^2 - 4Lx + 6L^2)$ 进行比较. 结构矩阵的条件数比固定梁的条件数更好还是更坏?

6. 给悬臂梁加上负载, 如步骤 4, 重新再计算. 如负载梁一样在同一图上绘制无负载梁.

7. 为进一步解释, 下面有一些想法: 若矩形截面变化为 $h = 10$ cm 和 b=5 cm, 梁变得更强还是更弱? 或者若截面是与矩形面积相同的圆形或环形又怎样呢? (半径为 r 的圆形截面的惯性面积矩是 $I = \pi r^4/4$, 内径为 r_1、外径为 r_2 的环形截面的 $I = \pi(r_2^4 - r_1^4)/4$.) 例如, 求出钢梁的惯性面积矩. 不同材料的杨氏模量被制作成表可供使用.

Euler-Berneoulli 梁是一个相对简单、经典的模型. 更多新的模型, 如 Timoshenko 梁, 考虑了更一般的弯曲——梁的截面可能不垂直于其主轴——因此常更精确.

2.6 共轭梯度法

当系数矩阵 $\boldsymbol{A}$ 有特殊的形状时, 存在求解 $\boldsymbol{Ax} = \boldsymbol{b}$ 的特殊的方法. 在 [7] 中发展起来的共轭梯度方法 (conjugate gradient method) 即为这些情形而设计, 通常应用于当 $\boldsymbol{A}$ 为对称正定矩阵时.

2.6.1 正定矩阵

定义 2.12 *$n \times n$ 矩阵 $\boldsymbol{A}$ 是对称的*(symmetric), *若 $\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{A}$. 矩阵 $\boldsymbol{A}$ 是正定的*(positive-definite), *若对所有的 $\boldsymbol{x} \neq \boldsymbol{0}, \boldsymbol{x}^{\mathrm{T}}\boldsymbol{Ax} > 0$.*

应用中的许多矩阵是对称的和正定的. 若 $\boldsymbol{A}$ 是对称的 (意味着它所有的特征值是实数), 那么当它所有的特征值都是正的时, $\boldsymbol{A}$ 就是正定的. 注意到由于非零向量 $\boldsymbol{x}$ 不可能满足 $\boldsymbol{Ax} = \boldsymbol{0}$, 所以一对称正定矩阵必须是非奇异的.

例 2.26 证明矩阵 $\boldsymbol{A} = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$ 是对称正定的.

显然 $\boldsymbol{A}$ 是对称的. 为证明其正定性, 可以求出它的特征值 (是 1 和 6) 或回到

定义:

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}=[x_1 \quad x_2]\begin{bmatrix}2 & 2\\ 2 & 5\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=2x_1^2+4x_1x_2+5x_2^2=2(x_1+x_2)^2+3x_2^2$$

这个表达式总是非负的且不为零, 除非 $x_2=0$ 及 $x_1+x_2=0$, 而这隐含着 $\boldsymbol{x}=0$.◀

例 2.27 证明对称矩阵 $\boldsymbol{A}=\begin{bmatrix}2 & 4\\ 4 & 5\end{bmatrix}$ 不是正定的.

由完成平方计算 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$:

$$\begin{aligned}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}&=[x_1 \quad x_2]\begin{bmatrix}2 & 4\\ 4 & 5\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=2x_1^2+8x_1x_2+5x_2^2=2(x_1^2+4x_1x_2)+5x_2^2\\ &=2(x_1+2x_2)^2-8x_2^2+5x_2^2=2(x_1+2x_2)^2-3x_2^2.\end{aligned}$$

例如, 令 $x_1=-2$ 和 $x_2=1$, 得到的结果小于零, 这与正定的定义矛盾. ◀

2.6.2 共轭梯度法

对称正定系统 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$, 可由下面的有限次循环求解:

共轭梯度法

$\boldsymbol{x}_0=\mathbf{0}$

$\boldsymbol{d}_0=\boldsymbol{r}_0=\boldsymbol{b}$

for $i=1,2,3,\cdots,n$

  if $\boldsymbol{r}_i=\mathbf{0}$, stop, end

  $\alpha_i=\dfrac{\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{r}_{i-1}}{\boldsymbol{d}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}}$

  $\boldsymbol{x}_i=\boldsymbol{x}_{i-1}+\alpha_i\boldsymbol{d}_{i-1}$

  $\boldsymbol{r}_i=\boldsymbol{r}_{i-1}-\alpha_i\boldsymbol{A}\boldsymbol{d}_{i-1}$

  $\beta_i=\dfrac{\boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{r}_i}{\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{r}_{i-1}}$

  $\boldsymbol{d}_i=\boldsymbol{r}_i+\beta_i\boldsymbol{d}_{i-1}$

end

解是 $\boldsymbol{x}_i$.

注意到由于 $\boldsymbol{A}$ 是正定的, 故 α_i 的分母是非零的. 在迭代的开始, 由定义 $\boldsymbol{A}\boldsymbol{x}_0+\boldsymbol{r}_0=\boldsymbol{b}$. 在迭代过程中, 注意到

$$\boldsymbol{A}\boldsymbol{x}_i+\boldsymbol{r}_i=\boldsymbol{A}\boldsymbol{x}_{i-1}+\alpha_i\boldsymbol{A}\boldsymbol{d}_{i-1}+\boldsymbol{r}_{i-1}-\alpha_i\boldsymbol{A}\boldsymbol{d}_{i-1}=\boldsymbol{A}\boldsymbol{x}_{i-1}+\boldsymbol{r}_{i-1}=\boldsymbol{b},$$

故由归纳法, 对所有的 i, $\boldsymbol{A}\boldsymbol{x}_i+\boldsymbol{r}_i=\boldsymbol{b}$ 成立, 因此 $\boldsymbol{r}_i=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_i$ 是第 i 步的残差 (见定义 2.4). 若 $\boldsymbol{r}_i=\mathbf{0}$, 则方程 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 已被解出, 解为 $\boldsymbol{x}_i$. 定理 2.13 证明了所有由共轭梯度迭代产生的 $\boldsymbol{r}_i$ 彼此相互正交. 由于它们是 n 维向量, $\boldsymbol{r}_i$ 中至多有 n 个

可以成对正交, 所以要么 $\boldsymbol{r}_n$, 要么先前的 $\boldsymbol{r}_i$ 必须为0. 所以至多 n 步后, 共轭梯度达到一个解. 本质上, 该方法是一个直接而非迭代方法.

定理 2.13 设 $\boldsymbol{A}$ 为对称正定的 $n\times n$ 矩阵, $\boldsymbol{b}$ 为一向量. 在共轭梯度方法中, 假设对 $i<n$ 有 $\boldsymbol{r}_i\neq\boldsymbol{0}$(若 $\boldsymbol{r}_i=\boldsymbol{0}$, 则方程已解), 则对每个 $1\leqslant i\leqslant n$,

(a) 下面 $\mathbf{R}^n$ 的 3 个子空间是相等的:

$$\langle\boldsymbol{x}_1,\cdots,\boldsymbol{x}_i\rangle=\langle\boldsymbol{r}_0,\cdots,\boldsymbol{r}_{i-1}\rangle=\langle\boldsymbol{d}_0,\cdots,\boldsymbol{d}_{i-1}\rangle.$$

(b) 残差是成对正交的: $\boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{r}_j=0$, 对于$j<i$.

(c) $\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_j=0$, 对于$j<i$.

证 (a) 由定义, $\boldsymbol{x}_i=\boldsymbol{x}_{i-1}+\alpha_i\boldsymbol{d}_{i-1}$, 由归纳法, 这隐含着 $\langle\boldsymbol{x}_1,\cdots,\boldsymbol{x}_i\rangle=\langle\boldsymbol{d}_0,\cdots,\boldsymbol{d}_{i-1}\rangle$. 利用 $\boldsymbol{d}_i=\boldsymbol{r}_i+\beta_i\boldsymbol{d}_{i-1}$ 可以证明 $\langle\boldsymbol{r}_0,\cdots,\boldsymbol{r}_{i-1}\rangle$ 与 $\langle\boldsymbol{d}_0,\cdots,\boldsymbol{d}_{i-1}\rangle$ 相等.

对 (b) 和 (c), 由归纳法继续进行下去. 当 $i=0$ 时, 它们显然成立, 没有什么要证明的. 对 $i\geqslant 1$, 在左边用 $\boldsymbol{r}_j^{\mathrm{T}}$ 乘上 $\boldsymbol{r}_i$ 的定义:

$$\boldsymbol{r}_j^{\mathrm{T}}\boldsymbol{r}_i=\boldsymbol{r}_j^{\mathrm{T}}\boldsymbol{r}_{i-1}-\frac{\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{r}_{i-1}}{\boldsymbol{d}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}}\boldsymbol{r}_j^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}. \tag{2.44}$$

若 $j\leqslant i-2$, 则由归纳假设 (b) 知 $\boldsymbol{r}_j^{\mathrm{T}}\boldsymbol{r}_{i-1}=0$. 由于 $\boldsymbol{r}_j$ 可以由 $\boldsymbol{d}_0,\cdots,\boldsymbol{d}_j$ 表示出来, 则由归纳假设 (c) 知 $\boldsymbol{r}_j^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}=0$. 另一方面, 若 $j=i-1$, 则由 (2.44) 又得 $\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{r}_i=0$, 这是因为由归纳假设, $\boldsymbol{d}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}=\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}+\beta_i\boldsymbol{d}_{i-2}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}=\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}$. 这就证明了 (b).

既然 $\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{r}_i=0$, 当 $j=i$ 时 (2.44) 说明

$$\frac{\boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{r}_i}{\boldsymbol{r}_{i-1}^{\mathrm{T}}\boldsymbol{r}_{i-1}}=-\frac{\boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}}{\boldsymbol{d}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}}.$$

这个式子与用 $\boldsymbol{d}_j^{\mathrm{T}}\boldsymbol{A}$ 在左边乘上 $\boldsymbol{d}_i$ 的定义一起得到

$$\boldsymbol{d}_j^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_i=\boldsymbol{d}_j^{\mathrm{T}}\boldsymbol{A}\boldsymbol{r}_i-\frac{\boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}}{\boldsymbol{d}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}}\boldsymbol{d}_j^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_{i-1}. \tag{2.45}$$

若 $j=i-1$, 利用 $\boldsymbol{A}$ 的对称性, 由 (2.45) 可得 $\boldsymbol{d}_{i-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{d}_i=0$. 若 $j\leqslant i-2$, 则 $\boldsymbol{A}\boldsymbol{d}_j=(\boldsymbol{r}_j-\boldsymbol{r}_{j+1})/\alpha_{j+1}$(由 $\boldsymbol{r}_i$ 的定义) 与 $\boldsymbol{r}_i$ 正交, 这证明了 (2.45) 的第一项为 0, 由归纳假设知第二项为 0, 这就完成了对 (c) 的论证.

例 2.28 用共轭梯度法求解 $\begin{bmatrix}2&2\\2&5\end{bmatrix}\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}6\\3\end{bmatrix}$.

由前面的算法, 我们有

$$x_0=\begin{bmatrix}0\\0\end{bmatrix},\quad r_0=d_0=\begin{bmatrix}6\\3\end{bmatrix},$$

$$\alpha_1=\frac{\begin{bmatrix}6\\3\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}6\\3\end{bmatrix}}{\begin{bmatrix}6\\3\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}2&2\\2&5\end{bmatrix}\begin{bmatrix}6\\3\end{bmatrix}}=\frac{45}{6\times18+3\times27}=\frac{5}{21},$$

$$x_1=\begin{bmatrix}0\\0\end{bmatrix}+\frac{5}{21}\begin{bmatrix}6\\3\end{bmatrix}=\begin{bmatrix}\frac{10}{7}\\\frac{5}{7}\end{bmatrix},$$

$$r_1=\begin{bmatrix}6\\3\end{bmatrix}-\frac{5}{21}\begin{bmatrix}18\\27\end{bmatrix}=12\begin{bmatrix}\frac{1}{7}\\-\frac{2}{7}\end{bmatrix},$$

$$\beta_1=\frac{r_1^{\mathrm{T}}r_1}{r_0^{\mathrm{T}}r_0}=\frac{144\times5/49}{36+9}=\frac{16}{49},$$

$$d_1=12\begin{bmatrix}\frac{1}{7}\\-\frac{2}{7}\end{bmatrix}+\frac{16}{49}\begin{bmatrix}6\\3\end{bmatrix}=\begin{bmatrix}\frac{180}{49}\\-\frac{120}{49}\end{bmatrix},$$

$$\alpha_2=\frac{\begin{bmatrix}\frac{12}{7}\\-\frac{24}{7}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\frac{12}{7}\\-\frac{24}{7}\end{bmatrix}}{\begin{bmatrix}\frac{180}{49}\\-\frac{120}{7}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}2&2\\2&5\end{bmatrix}\begin{bmatrix}\frac{180}{49}\\-\frac{120}{7}\end{bmatrix}}=\frac{7}{10},$$

$$x_2=\begin{bmatrix}\frac{10}{7}\\\frac{5}{7}\end{bmatrix}+\frac{7}{10}\begin{bmatrix}\frac{180}{49}\\-\frac{120}{49}\end{bmatrix}=\begin{bmatrix}4\\-1\end{bmatrix},$$

$$r_2=12\begin{bmatrix}\frac{1}{7}\\-\frac{2}{7}\end{bmatrix}-\frac{7}{10}\begin{bmatrix}2&2\\2&5\end{bmatrix}\begin{bmatrix}\frac{180}{49}\\-\frac{120}{49}\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}.$$

因 $r_2=b-Ax_2=0$, 故解为 $x_2=[4,-1]$. ◀

在例 2.28 中, 注意到定理 2.13 保证了 r_1 与 r_0 正交, 这个事实是共轭梯度法成功的关键：每个新的残差 r_i 与前面所有的 r_i 都正交. 若有一个 r_i 变为$\mathbf{0}$, 则 $Ax_i=b$ 且 x_i 就是解. 若不是, 经过循环 n 步后, r_n 与由 n 个成对正交向量 $r_0,\cdots,r_{n-1}$ 所张成的空间正交, 而该空间必定是 $\mathbf{R}^n$, 故 r_n 必为零向量, 且 $Ax_n=b$.

共轭梯度法在某些方面比高斯消去法更简单. 比如说, 编写程序显得更加简单——无需顾虑行运算, 如在高斯消去法中一样没有 3 次循环. 这两种方法都是直接方法, 它们都在有限步内达到理论上的正确解. 故有两个问题须考虑：为什么共轭梯度法并不优于高斯消去法, 为什么共轭梯度法经常被当作是一个迭代方法?

我们从运算计数开始回答这两个问题. 执行完循环需要一次矩阵与向量的乘积 ($\boldsymbol{A}\boldsymbol{d}_{n-1}$) 与几次额外的点积. 仅矩阵与向量的乘积每一步就需 n^2 次乘法 (同时有相同次数的加法), 经 n 步后, 总计有 n^3 次乘法. 与高斯消去法 $\frac{n^3}{3}$ 次计数相比, 这是 3 倍的工作量, 太大了.

若 $\boldsymbol{A}$ 是稀疏的, 则情形就不一样了. 假设 n 太大使得就高斯消去法的 $\frac{n^3}{3}$ 次运算量而言已不可行, 则高斯消去法必须执行完毕方可给出解 $\boldsymbol{x}$, 而共轭梯度法在每步可给出一个近似解 $\boldsymbol{x}_i$.

残差的欧氏长度每步都在减少, 故至少按欧氏度量而言, 每一步 $\boldsymbol{A}\boldsymbol{x}_i$ 越来越接近 $\boldsymbol{b}$. 因此, 依靠监控 $\boldsymbol{r}_i$, 可以求出一个足够好的解以避免完成所有的 n 步. 在这种情况下, 共轭梯度法变得与迭代方法难以区分了.

当 $\boldsymbol{A}$ 是一个病态矩阵时, 由于这种方法对舍入误差累积的敏感性, 所以该方法在发现不久后即受到冷落. 事实上, 对病态矩阵, 共轭梯度法不如带有部分主元的高斯消去法. 现今, 利用**预优法**(preconditioning) 解决了这个障碍, 预优法实质上是把问题转变为求解一个更好条件的矩阵系统, 这样就可以应用共轭梯度法了. 参见 [8] 以获取更多的信息.

该方法的名称源于共轭梯度法真正所做到的: 沿着 n 维的一个二次抛物面的斜率下滑, 名称的 "梯度" 部分意为靠使用微积分寻找最速下降方向, "共轭" 的意思是并不是每一步都与另一步正交, 但至少残差 $\boldsymbol{r}_i$ 是这样的. 该方法的几何细节与其形成背景是令人感兴趣的, 但超出了本书的范围. 最早的文章 [7] 给出了完整的描述.

例 2.29 应用共轭梯度方法求解 $n = 100\ 000$ 的系统 (2.38).

经过共轭梯度法 20 步后, 按向量无穷范数, 计算解 $\boldsymbol{x}$ 与真解 $(1,\cdots,1)$ 之间的差小于 10^{-9}. 在一台个人电脑上, 总的运算时间不到 1 秒钟. ◀

习题 2.6

1. 把 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$ 表示成平方和的形式, 以证明下列矩阵是对称正定的:

(a) $\begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 3 \\ 3 & 10 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$.

2. 找出一个向量 $\boldsymbol{x} \neq \boldsymbol{0}$ 使得 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \leqslant 0$, 以证明下列对称矩阵不是正定的.

(a) $\begin{bmatrix} 1 & 0 \\ 0 & -3 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix}$; (d) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$.

3. 人工执行共轭梯度法求解下列问题:

(a) $\begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$.

4. 人工执行共轭梯度法求解下列问题：

(a) $\begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$; (b) $\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -3 \\ 3 \end{bmatrix}$.

5. 证明若 $d > 4$, 则矩阵 $\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 2 & d \end{bmatrix}$ 是正定的.

6. 对一般数量情形 $\boldsymbol{Ax} = \boldsymbol{b}$ 执行共轭梯度迭代, 其中 $\boldsymbol{A}$ 是 1×1 矩阵, 求出 $\alpha_1, \boldsymbol{x}_1$ 并证明 $\boldsymbol{r}_1 = \boldsymbol{0}$ 及 $\boldsymbol{Ax}_1 = \boldsymbol{b}$.

7. 求出所有的数 d 使得 $\boldsymbol{A} = \begin{bmatrix} 1 & -2 \\ -2 & d \end{bmatrix}$ 是正定的.

8. 求出所有的数 d 使得 $\boldsymbol{A} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & 1 \\ 0 & 1 & d \end{bmatrix}$ 是正定的.

计算机问题 2.6

1. 编写共轭梯度方法的 MATLAB 程序, 并用它求解下列方程组：

(a) $\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

2. 利用共轭梯度法的 MATLAB 程序, 求解下列方程组：

(a) $\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & 1 \\ 0 & 1 & 5 \end{bmatrix}\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 3 \\ -3 \\ 4 \end{bmatrix}$.

3. 用共轭梯度法对 (a) $n = 4$; (b) $n = 8$ 求解方程组 $\boldsymbol{Hx} = \boldsymbol{b}$, 其中 $\boldsymbol{H}$ 是 $n \times n$ 的 Hilbert 矩阵, $\boldsymbol{b}$ 是分量皆为 1 的向量.

4. 用共轭梯度法对 (a) $n = 6$;(b) $n = 12$ 求解例 2.25 的稀疏问题.

5. 用共轭梯度法对 $n = 100, n = 1\ 000$ 与 $n = 10\ 000$ 求解例 2.25. 给出最终残差的大小及所需的步数.

2.7 非线性方程组系统

第 1 章讲述了求解含一个未知量的单个方程 (通常是非线性的) 的方法. 第 2 章至今, 我们学习了方程组的求解方法, 但需要方程组是线性的. 非线性与 “多于一个方程” 结合在一起使难度增加很多. 本节描述了求解非线性方程组的 Newton 方法及其变形.

2.7.1 多变量 Newton 方法

单变量 Newton 方法

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

给出了多变量 Newton 方法的主要特征. 这两种方法都源于由 Taylor 展开式提供的线性近似. 例如, 设

$$\begin{cases} f_1(u,v,w)=0, \\ f_2(u,v,w)=0, \\ f_3(u,v,w)=0 \end{cases}$$

为 3 个含 3 个未知量 u, v, w 的非线性方程. 定义向量值函数 $\boldsymbol{F}(u,v,w)=(f_1,f_2,f_3)$, 用 $\boldsymbol{F}(\boldsymbol{x})=\boldsymbol{0}$ 表示问题 (2.46), 其中 $\boldsymbol{x}=(u,v,w)$.

类似于单变量情形里的导数 f', 可以定义Jacobi **矩阵**为

$$\mathrm{D}\boldsymbol{F}(\boldsymbol{x})=\begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} & \frac{\partial f_1}{\partial w} \\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial w} \\ \frac{\partial f_3}{\partial u} & \frac{\partial f_3}{\partial v} & \frac{\partial f_3}{\partial w} \end{bmatrix}.$$

在 $\boldsymbol{x}_0$ 点附近, 向量值函数 (vector-valued function) 的 Taylor 展开式是

$$\boldsymbol{F}(\boldsymbol{x})=\boldsymbol{F}(\boldsymbol{x}_0)+\mathrm{D}\boldsymbol{F}(\boldsymbol{x}_0)\cdot(\boldsymbol{x}-\boldsymbol{x}_0)+O(\boldsymbol{x}-\boldsymbol{x}_0)^2$$

例如, 在 $x_0=(0,0)$ 附近 $\boldsymbol{F}(u,v)=(\mathrm{e}^{u+v},\sin u)$ 的线性展开式是

$$\begin{aligned} F(x) &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} e^0 & e^0 \\ \cos 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + O(x)^2 \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} u+v \\ u \end{bmatrix} + O(x)^2. \end{aligned}$$

Newton 方法基于忽略了 $O(h^2)$ 项的一个线性近似. 正如在一维情形, 设 $\boldsymbol{x}=\boldsymbol{r}$ 为根, $\boldsymbol{x}_0$ 为当前估计, 则

$$\boldsymbol{0}=\boldsymbol{F}(\boldsymbol{r})\approx\boldsymbol{F}(\boldsymbol{x}_0)+\mathrm{D}\boldsymbol{F}(\boldsymbol{x}_0)\cdot(\boldsymbol{r}-\boldsymbol{x}_0),$$

或

$$-\mathrm{D}\boldsymbol{F}(\boldsymbol{x}_0)^{-1}\boldsymbol{F}(\boldsymbol{x}_0)\approx\boldsymbol{r}-\boldsymbol{x}_0. \tag{2.46}$$

所以, 根的一个更好的近似来源于求解 (2.46) 得到的 r.

多变量 Newton 方法

$$\begin{aligned} &\boldsymbol{x}_0 = \text{初始向量}, \\ &\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-(\mathrm{D}\boldsymbol{F}(\boldsymbol{x}_k))^{-1}\boldsymbol{F}(\boldsymbol{x}_k), \qquad k=0,1,2,\cdots. \end{aligned}$$

由于从计算的角度而言, 求逆是难以负担的, 所以我们用一个技巧来避免. 在每一步, 不用按照前述的定义, 设 $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\boldsymbol{s}$, 其中 $\boldsymbol{s}$ 为 $\mathrm{D}\boldsymbol{F}(\boldsymbol{x}_k)\boldsymbol{s}=\boldsymbol{F}(\boldsymbol{x}_k)$ 的

解. 现在, 仅需高斯消去法 ($\frac{n^3}{3}$ 次乘法) 执行一步而不用计算求逆 (大约 3 倍多运算量). 因此, 多变量 Newton 方法的迭代步骤为

$$\begin{cases} \mathrm{D}\boldsymbol{F}(\boldsymbol{x}_k)\boldsymbol{s} = -\boldsymbol{F}(\boldsymbol{x}_k), \\ \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}. \end{cases} \tag{2.47}$$

例 2.30 取初始估计 (1,2), 用 Newton 方法求解方程组

$$\begin{cases} v - u^3 = 0, \\ u^2 + v^2 - 1 = 0. \end{cases}$$

图 2-4 表示出了 $f_1(u,v) = v - u^3$ 和 $f_2(u,v) = u^2 + v^2 - 1$ 都为 0 的集合, 以及它们对应于方程组解的两个交点. Jacobi 矩阵是

$$\mathrm{D}\boldsymbol{F}(u,v) = \begin{bmatrix} -3u^2 & 1 \\ 2u & 2v \end{bmatrix}.$$

利用初始点 $x_0 = (1,2)$, 在第一步我们必须求解矩阵方程 (2.47):

$$\begin{bmatrix} -3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

解为 $\boldsymbol{s} = (0,-1)$, 故第一步迭代产生 $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s} = (1,1)$, 第二步需要求解

$$\begin{bmatrix} -3 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

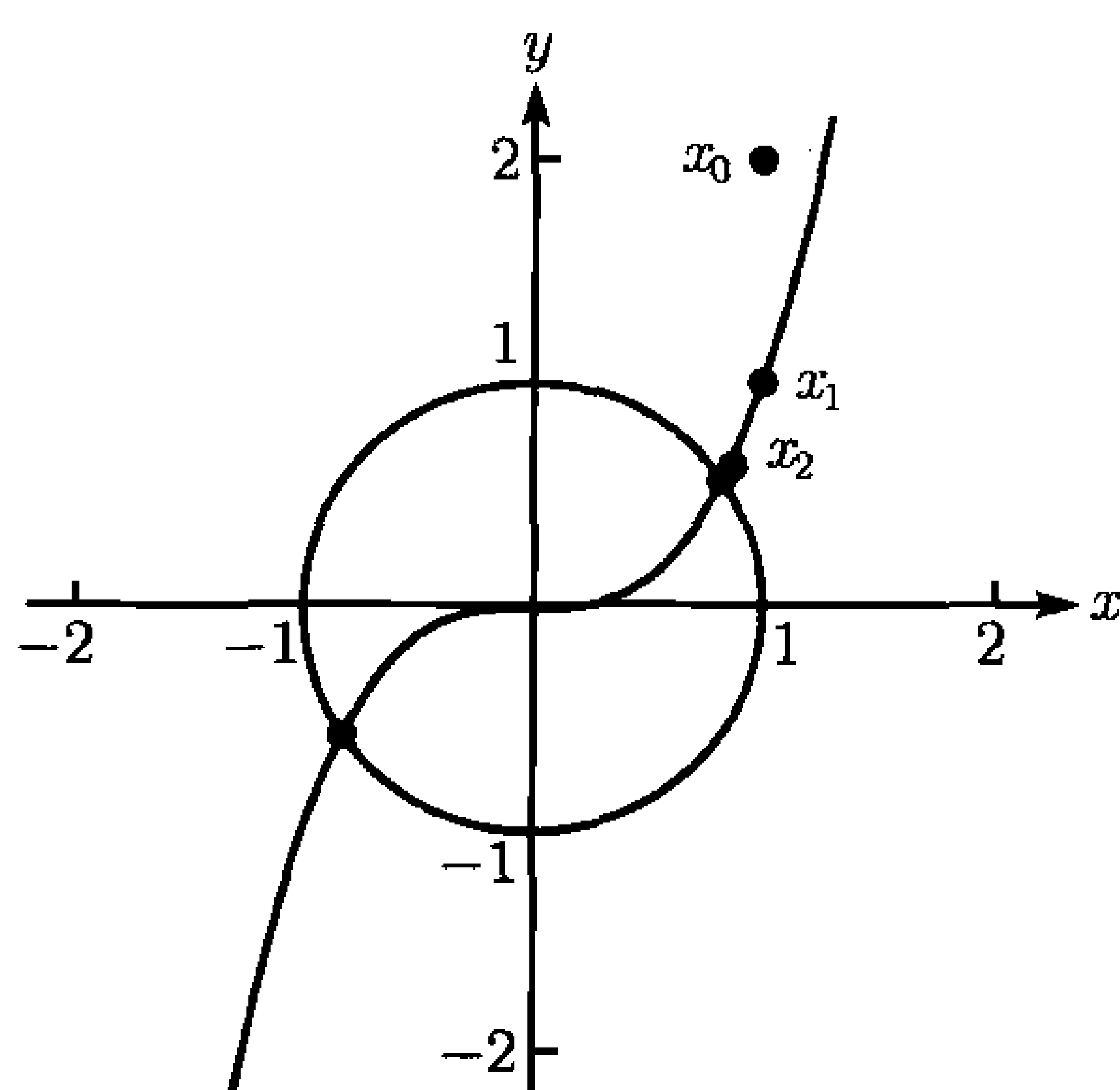


图 2-4 例 2.30 的 Newton 方法: 两个根是实心点. Newton 方法生成的空心点收敛到近似解 (0.826 0, 0.563 6)

解是 $\boldsymbol{s} = (-1/8, -3/8), \boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{s} = (7/8, 5/8)$. 两个迭代如图 2-4 所示. 进一步迭代得到表 2-3.

表 2-3

步骤	u	v
0	1.000 000 000 000 00	2.000 000 000 000 00
1	1.000 000 000 000 00	1.000 000 000 000 00
2	0.875 000 000 000 00	0.625 000 000 000 00
3	0.829 036 348 267 12	0.564 349 112 426 04
4	0.826 040 108 170 65	0.563 619 773 502 84
5	0.826 031 357 732 41	0.563 624 162 131 63
6	0.826 031 357 654 19	0.563 624 162 161 26
7	0.826 031 357 654 19	0.563 624 162 161 26

在输出序列中出现了显然的代表二次收敛性特征的准确小数位的加倍. 方程组的对称性说明若 (u,v) 是一个解, 则 $(-u,-v)$ 也是解, 这在图 2-4 中是明显的. 取一个邻近的初始估计, 应用 Newton 方法可求出第二个解. ◀

例 2.31 用 Newton 方法求出方程组

$$\begin{cases} f_1(u,v) = 6u^3 + uv - 3v^3 - 4 = 0, \\ f_2(u,v) = u^2 - 18uv^2 + 16v^3 + 1 = 0 \end{cases}$$

的解. 注意到 $(u,v) = (1,1)$ 是一个解. 事实表明, 还有另外两个解. Jacobi 矩阵是

$$\mathrm{D}\boldsymbol{F}(u,v) = \begin{bmatrix} 18u^2 + v & u - 9v^2 \\ 2u - 18v^2 & -36uv + 48v^2 \end{bmatrix}.$$

正如在一维的情形中一样, 其解由依赖于初始估计的 Newton 方法求出. 利用初始点 $(u_0, v_0) = (2,2)$, 迭代前述的公式得到表 2-4. 另外的初始向量导出其他两个根, 近似为 $(0.865\,939, 0.462\,168)$ 和 $(0.886\,809, -0.294\,007)$. 见计算机问题 2.

表 2-4

步骤	u	v
0	2.000 000 000 000 00	2.000 000 000 000 00
1	1.372 580 645 161 29	1.340 322 580 645 16
2	1.078 386 812 004 43	1.053 801 232 649 84
3	1.005 349 688 965 20	1.002 692 618 715 39
4	1.000 033 678 665 06	1.000 022 437 720 10
5	1.000 000 001 119 57	1.000 000 000 578 94
6	1.000 000 000 000 00	1.000 000 000 000 00
7	1.000 000 000 000 00	1.000 000 000 000 00

若能算出 Jacobi 矩阵, 则 Newton 方法是一个好的选择. 如若不然, 另一最好的选择是 Broyden 方法, 这是 2.7.2 节的主题. ◀

2.7.2 Broyden 方法

求解含一个未知量的一个方程的 Newton 方法需要知道导数. 对割线法的讨论进一步发展了第 1 章中的这种方法, 当导数不易获得或计算太费力时可使用割线法.

既然有了求解非线性方程组 $\boldsymbol{F}(\boldsymbol{x})=\mathbf{0}$ 的 Newton 方法的一种形式, 我们面临着相同的问题: 如果 Jacobi 矩阵 $\mathrm{D}\boldsymbol{F}$ 不能得到该如何呢? 尽管 Newton 方法不能简单地扩展为求解方程组的割线方法, 但 Broyden[2] 提出了一种通常情况下可被视为近似完美的方法 (next best method).

设 $\boldsymbol{A}_{i-1}$ 是在步骤 $i-1$ 得到的 Jacobi 矩阵的最好近似, 并已用于产生

$$\boldsymbol{x}_i=\boldsymbol{x}_{i-1}\boldsymbol{A}_{i-1}^{-1}\boldsymbol{F}(\boldsymbol{x}_{i-1}). \tag{2.48}$$

为下一步把 $\boldsymbol{A}_{i-1}$ 校正为 $\boldsymbol{A}_i$, 我们宁愿关注 Jacobi 矩阵 $\mathrm{D}\boldsymbol{F}$ 的导数方面, 满足

$$\boldsymbol{A}_i\boldsymbol{\delta}_i\approx\Delta_i, \tag{2.49}$$

其中 $\boldsymbol{\delta}_i=\boldsymbol{x}_i-\boldsymbol{x}_{i-1},\Delta_i=f(\boldsymbol{x}_i)-f(\boldsymbol{x}_{i-1})$. 另一方面, 对 $\boldsymbol{\delta}_i$ 的正交补, 我们没有新的信息. 因此, 我们要求

$$\boldsymbol{A}_i\boldsymbol{w}=\boldsymbol{A}_{i-1}\boldsymbol{w} \tag{2.50}$$

对每个 $\boldsymbol{w}$ 满足 $\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{w}=\mathbf{0}$. 同时满足 (2.49) 与 (2.50) 的矩阵是

$$\boldsymbol{A}_i=\boldsymbol{A}_{i-1}+\frac{(\Delta_i-\boldsymbol{A}_{i-1}\boldsymbol{\delta}_i)\boldsymbol{\delta}_i^{\mathrm{T}}}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}, \tag{2.51}$$

此时 (2.49) 变成了

$$\boldsymbol{A}_i\boldsymbol{\delta}_i=\boldsymbol{A}_{i-1}\boldsymbol{\delta}_i+\frac{(\Delta_i-\boldsymbol{A}_{i-1}\boldsymbol{\delta}_i)\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}=\Delta_i.$$

(2.50) 变成了

$$\boldsymbol{A}_i\boldsymbol{w}=\boldsymbol{A}_{i-1}\boldsymbol{w}+\frac{(\Delta_i-\boldsymbol{A}_{i-1}\boldsymbol{\delta}_i)\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{w}}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}=\boldsymbol{A}_{i-1}\boldsymbol{w}.$$

在用 (2.51) 校正近似的 Jacobi 矩阵时, Broyden 方法利用 Newton 方法步骤 (2.48) 改善当前估计. 总而言之, 取两个初始估计 x_0,x_1 及一个初始近似 Jacobi 矩阵 $\boldsymbol{A}_0$ 开始算法, 若没有更好的选择, $\boldsymbol{A}_0$ 可选为单位矩阵.

Broyden 方法 I

$\boldsymbol{x}_0,\boldsymbol{x}_1=$ 初始向量

$\boldsymbol{A}_0=$ 初始矩阵

for $i=1,2,3\cdots$

$\quad\boldsymbol{A}_i=\boldsymbol{A}_{i-1}+\frac{(\Delta\boldsymbol{\delta}_i-\boldsymbol{A}_{i-1}\boldsymbol{\delta}_i)\boldsymbol{\delta}_i^{\mathrm{T}}}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}$

$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{A}_i^{-1}\boldsymbol{F}(\boldsymbol{x}_i)$

end

其中 $\boldsymbol{\delta}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1}$ 并且 $\Delta_i = \boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}_{i-1})$.

注意到正如对 Newton 方法那样，我们通过求解 $\boldsymbol{A}_{i-1}\boldsymbol{\delta}_i = \boldsymbol{F}(\boldsymbol{x}_{i-1})$ 以完成类似于 Newton 方法的步骤. 尽管这意味着由于必须求解一个可能稠密的矩阵方程而对很多个方程进行的算法可能会很慢，但有一个技巧，使用 QR 分解 (在第 4 章中定义的) 可使复杂性减少至 $O(n^2)$. 同 Newton 方法一样，并不能保证 Broyden 方法收敛到解.

第二种实现 Broyden 方法的途径依赖于这个事实：(2.51) 把 $\boldsymbol{A}_i$ 定义为 $\boldsymbol{A}_{i-1}$ 的一个修正. 设 $u = (\Delta_i - \boldsymbol{A}_{i-1}\boldsymbol{\delta}_i/(\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i), \boldsymbol{v} = \boldsymbol{\delta}_i$, 则 $\boldsymbol{A}_i = \boldsymbol{A}_{i-1} + u\boldsymbol{v}^{\mathrm{T}}$. 由于我们只对 $\boldsymbol{A}_i^{-1}$ 感兴趣，故可利用 Sherman-Morrison 公式 (附录 A) 得到直接公式

$$\begin{aligned}
\boldsymbol{A}_i^{-1} &= (\boldsymbol{A}_{i-1} + u\boldsymbol{v}^{\mathrm{T}})^{-1} = \boldsymbol{A}_{i-1}^{-1} - \frac{\boldsymbol{A}_{i-1}^{-1}u\boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}_{i-1}^{-1}}{1 + \boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}_{i-1}^{-1}u} \\
&= \boldsymbol{A}_{i-1}^{-1} - \frac{\boldsymbol{A}_{i-1}^{-1}\left(\frac{\Delta_i - \boldsymbol{A}_{i-1}\boldsymbol{\delta}_i}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}\right)\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{A}_{i-1}^{-1}}{1 + \boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{A}_{i-1}^{-1}\left(\frac{\Delta_i - \boldsymbol{A}_{i-1}\boldsymbol{\delta}_i}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{\delta}_i}\right)} \\
&= \boldsymbol{A}_{i-1}^{-1} + \frac{(\boldsymbol{\delta}_i - \boldsymbol{A}_{i-1}^{-1}\boldsymbol{\delta}_i)\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{A}_{i-1}^{-1}}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{A}_{i-1}^{-1}\Delta_i}.
\end{aligned}$$

使用 Sherman-Morrison 公式可以得到下面的便利，即 $\boldsymbol{A}_i$ 并不需要计算；可以通过逆 $\boldsymbol{B}_i \equiv \boldsymbol{A}_i^{-1}$ 进行校正. 改写 (2.51) 和 (2.48) 得到一个叫 Broyden 方法 II 的算法.

Broyden 方法 II

$\boldsymbol{x}_0, \boldsymbol{x}_1 =$ 初始向量

$\boldsymbol{B}_0 =$ 初始矩阵

for $i = 1, 2, 3 \cdots$

$\boldsymbol{B}_i = \boldsymbol{B}_{i-1} + \frac{(\boldsymbol{\delta}_i - \boldsymbol{B}_{i-1}\Delta_i)\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{B}_{i-1}}{\boldsymbol{\delta}_i^{\mathrm{T}}\boldsymbol{B}_{i-1}\Delta_i}$

$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \boldsymbol{B}_i\boldsymbol{F}(\boldsymbol{x}_i)$

end

其中 $\boldsymbol{\delta}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1}$ 并且 $\Delta_i = \boldsymbol{F}(\boldsymbol{x}_i) - \boldsymbol{F}(\boldsymbol{x}_{i-1})$.

开始时，需要两个初始向量估计 $\boldsymbol{x}_0, \boldsymbol{x}_1$ 与一个初始矩阵估计 $\boldsymbol{B}_0$. 若不能计算导数，可选择 $\boldsymbol{B}_0 = \boldsymbol{I}$.

Broyden II 的不足之处是不易得到在某些应用场合所需的 Jacobi 矩阵的估计. 矩阵 $\boldsymbol{B}_i$ 是 Jacobi 矩阵逆的一个估计. 另一方面，Broyden I 明了 $\boldsymbol{A}_i$，它估计了 Jacobi 矩阵. 因此，在一些范围内，Broyden I 和 II 被分别看作是 “优良的 Broyden”

与"较差的 Broyden".

Broyden 方法的两种形式都是超线性收敛的 (对单根而言), 比 Newton 方法的二次收敛性慢. 若得到了求 Jacobi 矩阵的公式, 使用 $DF(\boldsymbol{x}_0)$ 的逆往往可以加快收敛. 那么对于初始矩阵 $\boldsymbol{B}_0$, Broyden 方法 II 的 MATLAB 程序如下:

```
% Program 2.3 Broyden's Method II
% Input: k = max steps, x0, x1 initial vectors
% Output: solution x
% Example usage: broyden2(10,[1;2],[2;1])
function x=broyden2(k,x0,x1) [n,m]=size(x0);
b=eye(n,n);             % initial b
for i=2:k
  del=x1-x0;delta=f(x1)-f(x0);
  b=b+(del-b*delta)*del'*b/(del'*b*delta);
  x=x1-b*f(x1);
  x0=x1;x1=x;
end

function y=f(x)
y=zeros(2,1);
y(1)=x(2)-x(1)^3;
y(2)=x(1)^2+x(2)^2-1;
```

Broyden 方法, 无论哪种形式, 在 Jacobi 矩阵不可能得到的情况下都是很有用的. 实例检验 7 中的挠度弯曲模型说明了这种情况的典型例子.

习题 2.7

1. 求出下列函数的 Jacobi 矩阵:

(a) $\boldsymbol{F}(u,v)=(u^3,uv^3)$; **(b)** $\boldsymbol{F}(u,v)=(\sin uv,\mathrm{e}^{uv})$;

(c) $\boldsymbol{F}(u,v)=(u^2+v^2-1,(u-1)^2+v^2-1)$; **(d)** $\boldsymbol{F}(u,v,w)=(u^2+v-w^2,\sin uvw,uvw^4)$.

2. 利用 Taylor 展开式求出 x_0 附近 $\boldsymbol{F}(\boldsymbol{x})$ 的线性近似 $\boldsymbol{L}(\boldsymbol{x})$:

(a) $\boldsymbol{F}(u,v)=(1+\mathrm{e}^{u+2v},\sin(u+v)),\boldsymbol{x}_0=(0,0)$;

(b) $\boldsymbol{F}(u,v)=(u+\mathrm{e}^{u-v},2u+v),\boldsymbol{x}_0=(1,1)$.

3. 在 uv 平面上画出两条曲线的草图, 并用简单的代数方法精确地求出所有解:

(a) $\begin{cases} u^2+v^2=1, \\ (u-1)^2+v^2=1; \end{cases}$ **(b)** $\begin{cases} u^2+4v^2=4, \\ 4u^2+v^2=4; \end{cases}$ **(c)** $\begin{cases} u^2-4v^2=4, \\ (u-1)^2+v^2=4. \end{cases}$

4. 对习题 3 中的方程组应用 Newton 方法两步, 取初始点 (1,1).

5. 对习题 3 中的方程组应用 Broyden I 两步, 取初始点 (1,1) 及 (1,2) 取 $\boldsymbol{A}_0=\boldsymbol{I}$.

6. 对习题 3 中的方程组应用 Broyden II 两步, 取初始点 (1,1) 及 (1,2) 取 $\boldsymbol{A}_0=\boldsymbol{I}$.

计算机问题 2.7

1. 取适当的初始点, 执行 Newton 方法, 求出所有的解. 用习题 3 进行检验以确保你的答案是正确的.

(a) $\begin{cases} u^2+v^2=1, \\ (u-1)^2+v^2=1; \end{cases}$ (b) $\begin{cases} u^2+4v^2=4, \\ 4u^2+v^2=4; \end{cases}$ (c) $\begin{cases} u^2-4v^2=4, \\ (u-1)^2+v^2=4. \end{cases}$

2. 利用 Newton 方法求出例 2.31 的 3 个解.

3. 利用 Newton 方法求出方程组 $u^3-v^3+u=0$ 和 $u^2+v^2=1$ 的两个解.

4. 取初始估计 $\boldsymbol{x}_0=(1,1)$ 和 $(1,2)$ 及 $\boldsymbol{A}_0=\boldsymbol{I}$, 用 Broyden I 求解习题 3 中的方程组. 给出尽可能精确的解及所需的步数.

5. 取初始估计 (1, 1) 和 (1, 2) 及 $\boldsymbol{B}_0=\boldsymbol{I}$, 用 Broydent II 求解习题 3 中的方程组. 给出尽可能精确的解及所需的步数.

软件与进一步阅读

数值线性代数领域已有许多优秀的著作, 包括 [12] 及综合性的论著 [5]. [3,14] 是这个书目中新增的两本. 有关迭代方法讨论的书目包括 [1,6,8,11,13,15,16,4].

LAPACK 是一个综合性的、不受版权限制的软件包, 它包含高质量的矩阵代数计算程序, 其中有 $\boldsymbol{Ax}=\boldsymbol{b}$ 的求解、矩阵分解及条件数估计等方法. 程序都经过了细心编写, 可移植到包含共享内存向量和并行处理器的现代计算机体系结构中.

LAPACK 的可移植性依赖于这个事实: 其算法的编写最大限度地发挥了基本线性代数子程序 (Basic Linear Algebra Subprogram, BLAS) 的用途. BLAS 是基本的矩阵/向量计算的集合, 它可以在特殊的机器和结构上进行优化处理. BLAS 大致上分成三个部分: 第 1 层, 需要像内积这样的 $O(n)$ 次运算; 第 2 层, 像矩阵/向量乘法这样需 $O(n^2)$ 次运算; 第 3 层, 包括复杂度为 $O(n^3)$ 的矩阵/矩阵相乘运算.

LAPACK 中利用 $\boldsymbol{PA}=\boldsymbol{LU}$ 分解, 按双精度求解 $\boldsymbol{Ax}=\boldsymbol{b}$ 的一般稠密矩阵的程序叫做 DGESV. 对稀疏和带状矩阵还有其他的形式, 可参见 www.netlib.org/lapack 以获得更多的细节. LAPACK 程序的实现也形成了 MATLAB 矩阵代数计算以及 IMSL 和 NAG 软件包的基础.

第3章 插 值

虽然多项式插值是一种古老的实践, 但是插值在重工业中的应用开始于 20 世纪的三次样条. 受到船舶制造和飞机工业的实践的启发, 先是在欧洲的雪铁龙工作的工程师 Paul de Casteljau 和雷诺的工程师 Pierre Bézier, 随后在美国通用汽车厂工作的其他人, 一起推动了现在称为三次样条和 Bézier 样条的建立.

尽管样条方法发源于汽车的空气动力学研究, 但现在已在包括计算机排版在内的许多应用中得到使用. 印刷业的革命是由两位施乐公司的工程师引起的, 他们在 1984 年创建了名为 Adobe 的公司, 并发布了 PostScript 语言. 它引起了苹果公司的 Steve Jobs 的注意, 当时他正在寻找控制新发明的激光打印机的方法. Bézier 样条采用了简单的方法使相同的数学曲线适应多种打印机分辨率.

实例检验 3.5 节后的实例检验 3 探索了 PostScript 如何把字母构造成 Bézier 样条.

表达数据的有效方法是进一步理解科学问题的基础. 其中最基础的, 通过多项式来近似数据是一种数据压缩行为. 假设点 (x,y) 取自给定的函数 $y=f(x)$ 或者取自实验, 这里 x 表示温度而 y 表示反应速度. 实数上的函数代表无穷多的信息量. 通过这组数据求多项式就是用可以在有限步内求出结果的规则来代替这种信息. 尽管期待这种多项式在新输入的 x 处准确地代表函数是不现实的, 但是它可以充分接近地求解实际问题.

本章介绍多项式插值和样条插值, 使用它们便于寻求通过给定数据点的函数.

3.1 数据和插值函数

如果一个函数通过一组数据点, 那么就称这个函数**插值**了这组数据点. 假设我们收集了一组数据点 (x,y), 譬如 (0, 1), (2, 2), (3, 4). 如图 3-1 所示, 有一条经过这 3 点的抛物线. 我们把这条抛物线称为经过这 3 点的二次插值多项式.

定义 3.1 如果 $P(x_i)=y_i(1\leqslant i\leqslant n)$, 那么函数 $y=P(x)$**插值**了数据点

$$(x_1,y_1),\cdots,(x_n,y_n).$$

注意, 要求 P 是一个函数, 即对每一个 x 值都有对应的单个值 y. 这在能被插值的数据点 $\{(x_i,y_i)\}$ 上加了一种限制——为了使函数经过这些数据点, x_i 必须都

是不同的. 关于 y_i 却没有这种限制.

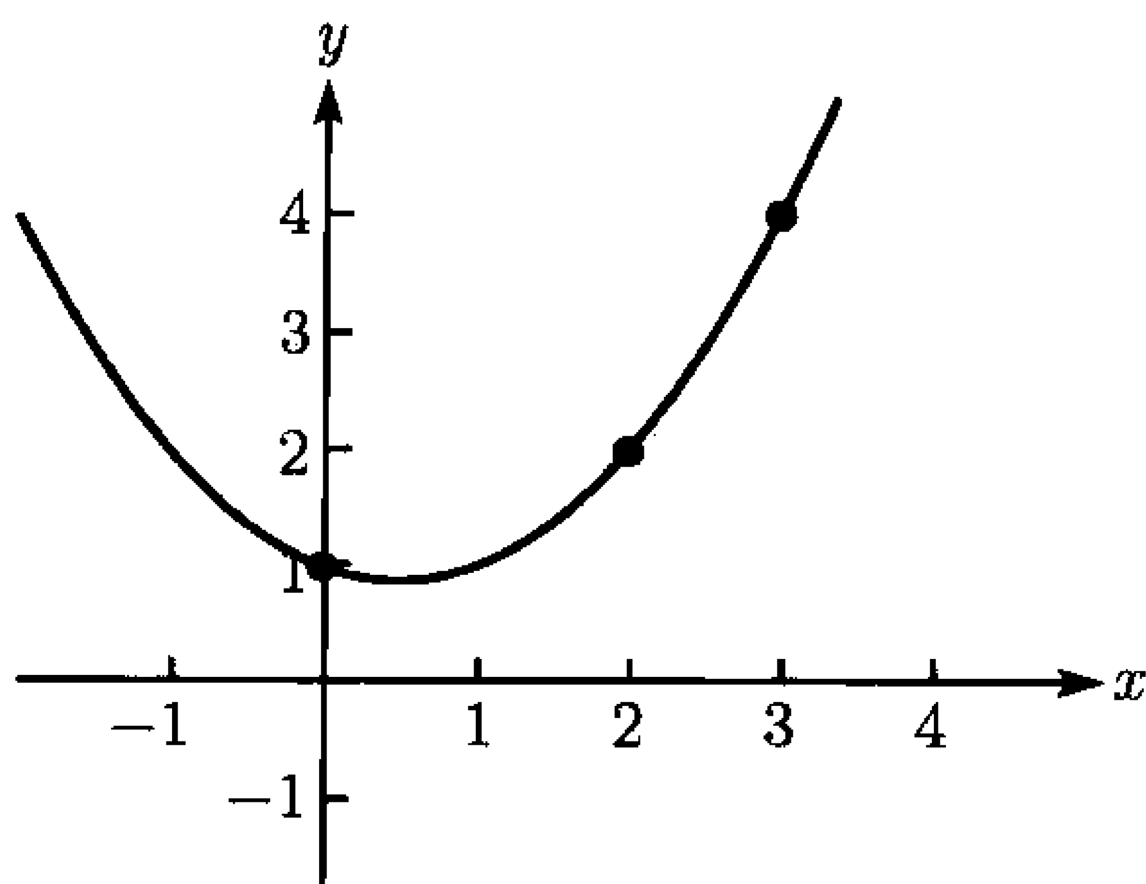


图 3-1 抛物线插值: 用函数 $P(x)=\frac{1}{2}x^2-\frac{1}{2}x+1$ 对点 (0, 1), (2, 2), (3, 4) 进行插值

亮点 复杂性

我们为什么用多项式? 多项式经常应用于插值是由于其简单明了的数学性质. 对于一组给定的点, 有一个简单的原理可以说明, 什么时候存在给定次数的插值多项式. 更重要的是, 在实际意义上, 多项式是数字计算机最基本的函数. 中央处理单元在硬件上对于浮点数的加法和乘法一般都有快速方法, 而这正是计算多项式所需要的运算. 可以用插值多项式来近似复杂函数, 这样使用这两种硬件运算就可以计算出它们.

作为开始, 我们将寻求插值多项式. 这种多项式永远都存在吗? 假设点的 x 坐标都不同, 则回答是肯定的. 不论给定多少点, 总存在经过所有点的某一多项式 $y=P(x)$. 本节证明关于插值多项式的这个事实以及其他若干事实.

插值与计算相反. 在多项式计算时 (譬如第 0 章中的嵌套乘法), 往往会给定一个多项式, 要求对给定的 x 值计算 y 值, 即计算落在曲线上的点. 多项式插值要求相反的过程: 给定这些点, 计算能生成它们的多项式.

3.1.1 Lagrange 插值

假设给定 n 个数据点 $(x_1,y_1),\cdots,(x_n,y_n)$, 我们想要求出一个插值多项式. 为了写出在这些点插值的次数是 $d=n-1$ 的多项式, 可以使用一个称为 Lagrange 插值公式的显式公式. 例如, 假设给定点 $(x_1,y_1),(x_2,y_2),(x_3,y_3)$, 那么多项式

$$P_2(x)=y_1\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}+y_2\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}+y_3\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} \quad (3.1)$$

就是关于这些点的 Lagrange **插值多项式**. 首先注意为什么所有这些点都落在这

条多项式曲线上. 当 x 用 x_1 代入时, 各项计算得 $y_1+0+0=y_1$. 当 x_1 代入后, 第二项和第三项的分子变成零, 而第一项的分子刚好等于分母, 因此结果为 y_1. 在代入 x_2 和 x_3 时也是类似的. 在用任何其他数代入 x 时, 我们几乎不能控制其结果. 但是, 我们的任务仅仅是在这 3 个点插值——这是我们关心的范围. 其次, 注意多项式 (3.1) 关于变量 x 是 2 次的.

例 3.1 为图 3-1 中的数据点 (0, 1), (2, 2), (3, 4) 寻找一个插值多项式.

代入 Lagrange 公式 (3.1) 得到

$$\begin{aligned} P_2(x) &= 1\frac{(x-2)(x-3)}{(0-2)(0-3)} + 2\frac{(x-0)(x-3)}{(2-0)(2-3)} + 4\frac{(x-0)(x-2)}{(3-0)(3-2)} \\ &= \frac{1}{6}(x^2-5x+6) + 2\left(-\frac{1}{2}\right)(x^2-3x) + 4\left(\frac{1}{3}\right)(x^2-2x) \\ &= \frac{1}{2}x^2 - \frac{1}{2}x + 1. \end{aligned}$$

检验得 $P_2(0)=1, P(2)=2, P_2(3)=4$. ◀

一般地, 假设给出 n 个点 $(x_1,y_1),\cdots,(x_n,y_n)$, 则对于 1 和 n 之间的每一个 k 可定义

$$L_k(x) = \frac{(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}.$$

L_k 的有趣性质是 $L_k(x_k)=1$ 而 $L_k(x_j)=0$, 其中 x_j 是其他任何数据点, 然后定义 $n-1$ 次多项式

$$P_{n-1}(x) = y_1L_1(x) + \cdots + y_nL_n(x).$$

这是 (3.1) 中多项式的直接一般化, 而且作用方式相同. 把 x_k 代替 x, 得到

$$P_{n-1}(x_k) = y_1L_1(x_k) + \cdots + y_nL_n(x_k) = 0 + \cdots + 0 + y_kL_k(x_k) + 0 + \cdots + 0 = y_k,$$

所以它的功能如所设计的.

我们已经构造了经过不同 x_i 的任何 n 个点的最高为 $n-1$ 次的多项式. 有趣的是, 它仅有一个.

定理 3.2 (多项式插值的主要定理) *设 $(x_1,y_1),\cdots,(x_n,y_n)$ 是平面上 x_i 互不相同的 n 个点, 那么存在一个而且仅存在一个次数小于等于 $n-1$ 次的多项式, 满足 $P(x_i)=y_i, i=1,\cdots,n$.*

证 存在性由 Lagrange 插值的显式公式得出. 为了证明仅有一个, 可先假设有两个, 譬如 $P(x)$ 及 $Q(x)$, 它们最多是 $n-1$ 次, 而且都插值所有 n 个点, 即假设, $P(x_1)=Q(x_1)=y_1, P(x_2)=Q(x_2)=y_2,\cdots,P(x_n)=Q(x_n)=y_n$. 现在定义

新的多项式 $H(x) = P(x) - Q(x)$. 显然, H 的次数也最多是 $n-1$ 次, 而且注意到 $0 = H(x_1) = H(x_2) = \cdots = H(x_n)$, 即 H 有 n 个不同的零点. 按照代数学基本定理, 一个 d 次多项式, 除了它恒等于零多项式, 最多可能有 d 个零点. 因此, H 是恒等于零多项式, 于是 $P(x) \equiv Q(x)$. 由此我们得到结论: 存在唯一的次数小于等于 $n-1$ 的多项式 $P(x)$ 插值于 n 个点 (x_i, y_i).

例 3.2 求插值于点 (0, 2), (1, 1), (2, 0), (3, −1) 的次数小于等于 3 的多项式. Lagrange 形式如下:

$$
\begin{aligned}
P(x) =& 2\frac{(x-1)(x-2)(x-3)}{(0-1)(0-2)(0-3)} + 1\frac{(x-0)(x-2)(x-3)}{(1-0)(1-2)(1-3)} \\
&+0\frac{(x-0)(x-1)(x-3)}{(2-0)(2-1)(2-3)} - 1\frac{(x-0)(x-1)(x-2)}{(3-0)(3-1)(3-2)} \\
=& -\frac{1}{3}(x^3 - 6x^2 + 11x - 6) + \frac{1}{2}(x^3 - 5x^2 + 6x) - \frac{1}{6}(x^3 - 3x^2 + 2x) \\
=& -x + 2.
\end{aligned}
$$

定理 3.2 说明, 恰好存在一个次数小于等于 3 的插值多项式, 但是它可能恰好是 3 次也可能不是 3 次. 在例 3.2 中, 数据点共线, 因此插值多项式是一次的. 定理 3.2 意味着不存在 2 次或 3 次的插值多项式. 或许没有抛物线或立方曲线可以经过 4 个共线的点在直觉上已经很明显, 但是这里才提供其原因. ◀

3.1.2 Newton 均差

如 3.1.1 节所述, 使用 Lagrange 插值方法, 可以构造出由定理 3.2 所预示的唯一的多项式. 它也是直观的, 只看一眼就可解释它为什么起作用. 然而很少用它来计算, 这是因为替代方法更具操作性而且是在计算上复杂程度更低的形式.

Newton 均差对写出插值多项式给出了特别简单的方式. 给定 n 个数据点, 其结果将是最多 $n-1$ 次的多项式, 这恰如 Lagrange 形式所具有的那样. 定理 3.2 表明它可能不外乎是与 Lagrange 插值多项式一样, 写成了更一般的形式.

均差的概念相当简单, 但是某些记号首先需要掌握. 把数据点看作由某个函数 f 给出, 并把数据点列成下表:

$$
\begin{array}{c|c}
x_1 & f(x_1) \\
x_2 & f(x_2) \\
\vdots & \vdots \\
x_n & f(x_n)
\end{array}
$$

现在定义都是实数的均差

$$f[x_k] = f(x_k),$$
$$f[x_k\ x_{k+1}] = \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k},$$

$$\begin{aligned} f[x_k\ x_{k+1}\ x_{k+2}] &= \frac{f[x_{k+1}\ x_{k+2}] - f[x_k\ x_{k+1}]}{x_{k+2} - x_k}, \\ f[x_k\ x_{k+1}\ x_{k+2}\ x_{k+3}] &= \frac{f[x_{k+1}\ x_{k+2}\ x_{k+3}] - f[x_k\ x_{k+1}\ x_{k+2}]}{x_{k+3} - x_k}, \end{aligned} \tag{3.2}$$

等等. 这些数是关于这些数据点的插值多项式的系数. 插值多项式由 **Newton 均差公式**给出:

$$\begin{aligned} P(x) = f[x_1] &+ f[x_1\ x_2](x - x_1) \\ &+ f[x_1\ x_2\ x_3](x - x_1)(x - x_2) \\ &+ f[x_1\ x_2\ x_3\ x_4](x - x_1)(x - x_2)(x - x_3) \\ &+ \cdots \\ &+ f[x_1 \cdots x_n](x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{3.3}$$

3.2.2 节给出了这个多项式插值 n 个数据点这个事实的证明. 注意, 均差公式给出了一个与嵌套多项式一样的插值多项式. 它自动准备好用有效的方法进行计算.

Newton 均差

给定 $\boldsymbol{x} = [x_1, \cdots, x_n], \boldsymbol{y} = [y_1, \cdots, y_n]$

for $j = 1, \cdots, n$

  $f[x_j] = y_j$

end

for $i = 2, \cdots, n$

  for $j = 1, \cdots, n+1-i$

$$f[x_j \cdots x_{j+i-1}] = (f[x_{j+1} \cdots x_{j+i-1}] - f[x_j \cdots x_{j+i-2}])/(x_{j+i-1} - x_j)$$

  end

end

插值多项式是

$$P(x) = \sum_{i=1}^{n} f[x_1 \cdots x_i](x - x_1) \cdots (x - x_{i-1})$$

Newton 均差的递归定义使得排列成为一个方便的表格. 对于 3 个点, 该表格的形式为

$$
\begin{array}{c|ccc}
x_1 & f[x_1] & & \\
 & & f[x_1\ x_2] & \\
x_2 & f[x_2] & & f[x_1\ x_2\ x_3] \\
 & & f[x_2\ x_3] & \\
x_3 & f[x_3] & &
\end{array}
$$

多项式 (3.8) 的系数可以从三角形的顶边读出.

例 3.3 利用均差求通过点 (0, 1), (2, 2), (3, 4) 的插值多项式.

应用均差的定义导出下表:

$$
\begin{array}{c|ccc}
0 & 1 & & \\
 & & \frac{1}{2} & \\
2 & 2 & & \frac{1}{2} \\
 & & 2 & \\
3 & 4 & &
\end{array}
$$

这张表格的计算如下: 在分开的列写下 x 和 y 坐标之后, 像 (3.2) 那样从左到右计算下一列. 例如:

$$\frac{2-1}{2-0}=\frac{1}{2}$$

$$\frac{2-\frac{1}{2}}{3-0}=\frac{1}{2}$$

$$\frac{4-2}{3-2}=2$$

在完成均差三角形后, 从这个三角形的顶边就能读出这个多项式的系数 $1, \frac{1}{2}, \frac{1}{2}$. 插值多项式可以写成

$$P(x)=1+\frac{1}{2}(x-0)+\frac{1}{2}(x-0)(x-2)$$

或者写成嵌套形式

$$P(x)=1+(x-0)\left(\frac{1}{2}+(x-2)\cdot\frac{1}{2}\right).$$

嵌套形式的基点 (见第 0 章) 是 $r_1=0$ 及 $r_2=2$. 另一方面, 我们可以做更多的代数, 从而把插值多项式写成

$$P(x)=1+\frac{1}{2}x+\frac{1}{2}x(x-2)=\frac{1}{2}x^2-\frac{1}{2}x+1,$$

它与前述 Lagrange 插值形式相匹配. ◀

利用均差方法, 在计算原来的插值多项式之后新加入的数据点可以很容易加上去.

例 3.4 在例 3.3 的表格中加入第 4 个数据点 (1, 0).

我们可以保留已做的计算, 并且仅需给三角形加上一条新的底边:

$$
\begin{array}{c|cccc}
0 & 1 & & & \\
 & & \frac{1}{2} & & \\
2 & 2 & & \frac{1}{2} & \\
 & & 2 & & -\frac{1}{2} \\
3 & 4 & & 0 & \\
 & & 2 & & \\
1 & 0 & & &
\end{array}
$$

结果是对原多项式 $P_2(x)$ 加上新的项. 从这个三角形的顶边读出, 我们看到新的三次插值多项式是

$$P_3(x) = 1 + \frac{1}{2}(x-0) + \frac{1}{2}(x-0)(x-2) - \frac{1}{2}(x-0)(x-2)(x-3).$$

注意 $P_3(x) = P_2(x) - \frac{1}{2}(x-0)(x-2)(x-3)$, 所以前一个多项式可以重新用作新的多项式的一部分. ◀

加入一个新的点到 Lagrange 公式所需要的额外工作与均差公式进行比较是很有趣的. 在加入一个新的点时, Lagrange 多项式必须重新开始, 以前的计算都不能用. 但是另一方面, 对于均差形式, 我们保留早先的工作, 只要把新的一项加到多项式. 因此, 均差方法具有 Lagrange 方法所缺少的 “实时更新” 的性质. 在习题 15 中比较了 Lagrange 型与均差型插值的运算次数.

例 3.5 用 Newton 均差公式, 求经过 (0, 2), (1, 1), (2, 0), (3, −1) 的插值多项式. 均差三角形是

$$
\begin{array}{c|cccc}
0 & 2 & & & \\
 & & -1 & & \\
1 & 1 & & 0 & \\
 & & -1 & & 0 \\
2 & 0 & & 0 & \\
 & & -1 & & \\
3 & -1 & & &
\end{array}
$$

读出系数, 我们求得次数小于等于 3 次的插值多项式是

$$P(x) = 2 + (-1)(x - 0) = 2 - x.$$

这与例 3.2 相一致, 但用了少得多的工作量. ◀

3.1.3 经过 n 个点的 d 次多项式有多少个

如果 $0 \leqslant d \leqslant n-1$, 那么多项式插值的主要定理 (即定理 3.2) 回答了这个问题. 给定 $n=3$ 个点 (0, 1), (2, 2), (3, 4), 存在一个次数小于等于 2 的插值多项式, 例 3.1 说明它是二次, 因此不存在零次或一次插值多项式经过这 3 个数据点.

有多少个三次多项式插值这 3 个相同的点? 从前面的讨论可知, 构造这种多项式的方法是明确的: 加上第 4 个点, 扩充 Newton 均差三角形给出新的最高次项系数, 在例 3.4 中加进了点 (1, 0). 结果得到的多项式是

$$P_3(x) = P_2(x) - \frac{1}{2}(x-0)(x-2)(x-3), \tag{3.4}$$

它既经过问题中的 3 个点, 还经过新的点 (1, 0), 因此至少有一个三次多项式经过原来的 3 个点 (0, 1), (2, 2), (3, 4).

当然, 我们有许多不同的方法选择第 4 个点. 例如, 如果我们保持相同的 $x_4 = 1$, 而简单地改变 y 让它不再等于 0, 由于函数在 x_4 处只能经过一个 y, 我们必定得到一个不同的三次插值多项式. 现在我们知道, 有无穷多个在 3 点 (x_1, y_1), (x_2, y_2), (x_3, y_3) 插值的多项式, 这是因为对于固定的 x_4, y_4 有无穷多种选取方式, 而每一种选取都给出了不同的多项式. 这种想法表明: 给定 x_i 互不相同的 n 个数据点 (x_i, y_i), 有无穷多个经过它们的 n 次多项式.

再次观察 (3.4) 式, 使人想到建立经过 3 个点的插值多项式的更直接的方法. 代替加上第 4 个点产生新的三次项系数, 为什么不就写一个新的系数? 结果还在原来 3 个点插值吗? 回答是肯定的, 这是因为 $P_2(x)$ 在原来的 3 个点插值, 而新的项在 x_1, x_2, x_3 取值为零. 所以, 实在不需要构造额外的 Newton 均差. 形式为

$$P_3(x) = P_2(x) + cx(x-2)(x-3)$$

(其中 $c \neq 0$) 的三次多项式将经过 (0, 1), (2, 2), (3, 4). 如在下例中叙述的, 这种技巧也将容易构造 (无穷多个) 关于 n 个给定数据点的次数大于等于 n 的多项式.

例 3.6 有多少个次数 $0 \leqslant d \leqslant 5$ 的多项式经过数据点 (−1, −5), (0, −1), (2, 1), (3, 11)?

Newton 均差三角形是

$$
\begin{array}{c|cccc}
-1 & -5 & & & \\
 & & 4 & & \\
0 & -1 & & -1 & \\
 & & 1 & & 1 \\
2 & 1 & & 3 & \\
 & & 10 & & \\
3 & 11 & & &
\end{array}
$$

所以没有零次、一次或二次插值多项式, 而唯一的三次插值多项式是

$$P_3(x) = -5 + 4(x+1) - (x+1)x + (x+1)x(x-2).$$

有无穷多个四次插值多项式

$$P_4(x) = P_3(x) + c_1(x+1)x(x-2)(x-3), c_1 \neq 0$$

以及无穷多个五次插值多项式

$$P_5(x) = P_3(x) + c_2(x+1)x^2(x-2)(x-3), c_2 \neq 0.$$ ◀

3.1.4 插值编码

计算系数的 MATLAB程序**newtdd.m**如下:

```
%Program 3.1 Newton Divided Difference Interpolation Method
%Computes coefficients of interpolating polynomial
%Input: x and y are vectors containing the x and y coordinates
%       of the n data points
%Output:coefficients c of interpolating polynomial in nested form
%Use with nest.m to evaluate interpolating polynomial
function c=newtdd(x,y,n)
for j=1:n
  v(j,1)=y(j);          % Fill in y column of Newton triangle
end
for i=2:n               % For column i,
  for j=1:n+1-i         % fill in column from top to bottom
    v(j,i)=(v(j+1,i-1)-v(j,i-1))/(x(j+i-1)-x(j));
  end
end
for i=1:n
  c(i)=v(1,i);          % Read along top of triangle
end                     % for output coefficients
```

这段程序可以应用于例 3.3 中的数据点, 以返回上面求得的系数 1, $\frac{1}{2}$, $\frac{1}{2}$. 可以把这些系数用于嵌套乘法程序中以计算在不同的 x 值处的插值多项式.

例如, MATLAB代码段

```
x0=[0 2 3];
y0=[1 2 4];
c=newtdd(x0,y0,3);
x=0:.01:4;
y=nest(2,c,x,x0);
plot(x0,y0,'o',x,y)
```

将得到如图 3-1 所示的多项式曲线.

现在有了求插值多项式的系数 (newtdd.m) 以及计算多项式(nest.m) 的 MATLAB 代码, 可以把它们放在一起建立一种多项式插值程序. 程序clickinterp.m 应用 MATLAB 的绘图能力在产生插值多项式时就把它绘制出来. 见图 3-2. MATLAB 的鼠标输入命令ginput用于简化数据输入.

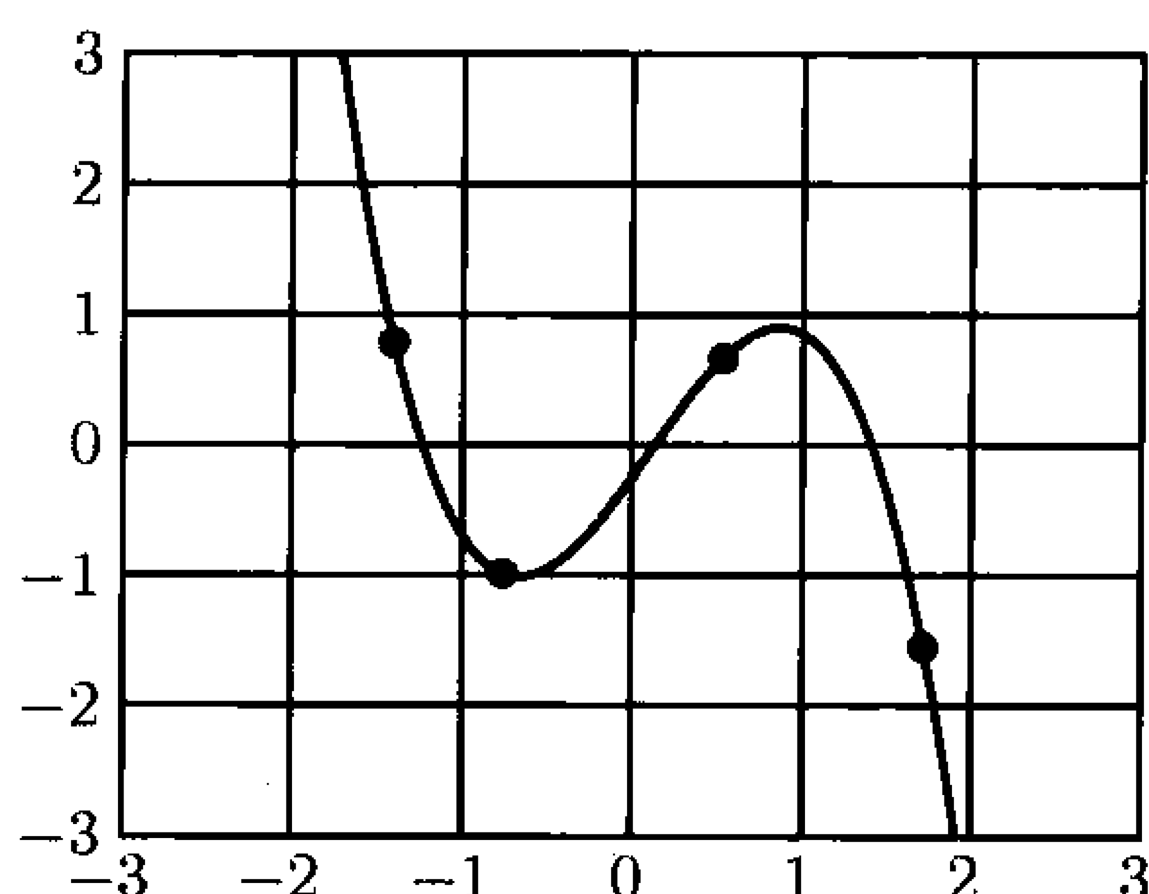


图 3-2 用鼠标输入的插值程序 3.2: 带 4 个输入数据点的 MATLAB代码 clickinterp.m 的屏幕截图

```
%Program 3.2. clickinterp.m  Polynomial Interpolation Program
%Left click in MATLAB figure window to locate data point.
%     Continue, to add more points.
%     Right-click to terminate program.
xl=-3;xr=3;yb=-3;yt=3;
plot([xl xr],[0 0],'k',[0 0],[yb yt],'k');grid on;
xlist=[];ylist=[];
button=1;k=0;        % initialize counter k
while(button ~= 3)   % if right click, terminate
  [xnew,ynew,button] = ginput(1);   % get one mouse click
  if button == 1     % current click is a left click
    k=k+1;           % k counts clicks
    xlist(k)=xnew; ylist(k)=ynew; % add new point to the list
    c=newtdd(xlist,ylist,k); % get interpolation coeffs
    x=xl:.01:xr;     % define x coordinates of curve
    y=nest(k-1,c,x,xlist); % get y coordinates of curve
    plot(xlist,ylist,'o',x,y,[xl xr],[0,0],'k',[0 0],[yb yt],'k');
    axis([xl xr yb yt]);grid on;
  end
end
```

亮点 压缩

这是我们第一次遇到数值分析中压缩 (compression) 的概念. 首先, 插值可以不看作压缩. 毕竟我们把 n 个点当作输入, 而提供插值多项式的 n 个系数作为输出. 那么压缩了什么?

把数据点想象成譬如曲线 $y=f(x)$ 上的许多点的代表. 由 n 个系数决定的 $n-1$ 次多项式就是 $f(x)$ 的一种 "压缩形式", 而在某些情形下, 为了计算目的可以用作 $f(x)$ 的相当简单的代表.

例如, 在按下计算器的正弦按钮时, 会发生什么? 计算器有硬件进行加法和乘法, 但是它如何计算一个数的正弦? 运算必须设法减少为恰好需要那些运算的多项式计算. 通过选择落在正弦曲线上的数据点, 就能计算出插值多项式并且作为正弦函数的压缩形式存储在计算器内.

这一类压缩是 "有损压缩", 即它会产生误差, 这是因为正弦函数并不是精确地等于多项式. 当函数 $f(x)$ 用插值多项式代替时会产生多大的误差, 这是 3.1.5 节的内容.

3.1.5 用近似多项式表示函数

多项式插值的主要用处是通过计算多项式来代替计算复杂函数, 而计算多项式仅仅包括诸如加法、减法及乘法这些基本的计算机运算. 把它理解成一种压缩的形式: 用某些简单而且可计算的东西来代替某些复杂的东西, 在这个过程中或许有我们将必须进行分析的某些精度损失. 我们从三角学的一个例子开始.

例 3.7 在 $[0,\frac{\pi}{2}]$ 中的 4 个等距点上插值函数 $f(x)=\sin x$.

我们在 $[0,\frac{\pi}{2}]$ 上压缩正弦函数. 在等距点取 4 个数据点并且形成均差三角形. 我们列出准确到 4 位的值.

x	f			
0	0.000 0			
		0.954 9		
$\frac{\pi}{6}$	0.500 0		−0.244 3	
		0.669 1		−0.113 9
$\frac{2\pi}{6}$	0.866 0		−0.423 2	
		0.255 9		
$\frac{3\pi}{6}$	1.000 0			

因此, 三次插值多项式是

$$P_3(x)=0+0.954\,9x-0.244\,3x\left(x-\frac{\pi}{6}\right)-0.113\,9x\left(x-\frac{\pi}{6}\right)\left(x-\frac{\pi}{3}\right)$$

$$= 0 + x\left(0.954\,9 + \left(x - \frac{\pi}{6}\right)\left(-0.244\,3 + \left(x - \frac{\pi}{3}\right)(-0.113\,9)\right)\right). \quad (3.5)$$

这个多项式与正弦函数一起在图 3-3 中给出. 在这种分辨率下, $P(x)$ 与 $\sin x$ 在区间 $[0, \frac{\pi}{2}]$ 中事实上没有区别. 我们已把正弦曲线无穷多的信息量压缩成存储几个系数以及在 (3.5) 中执行 3 次加法和 3 次乘法了. ◀

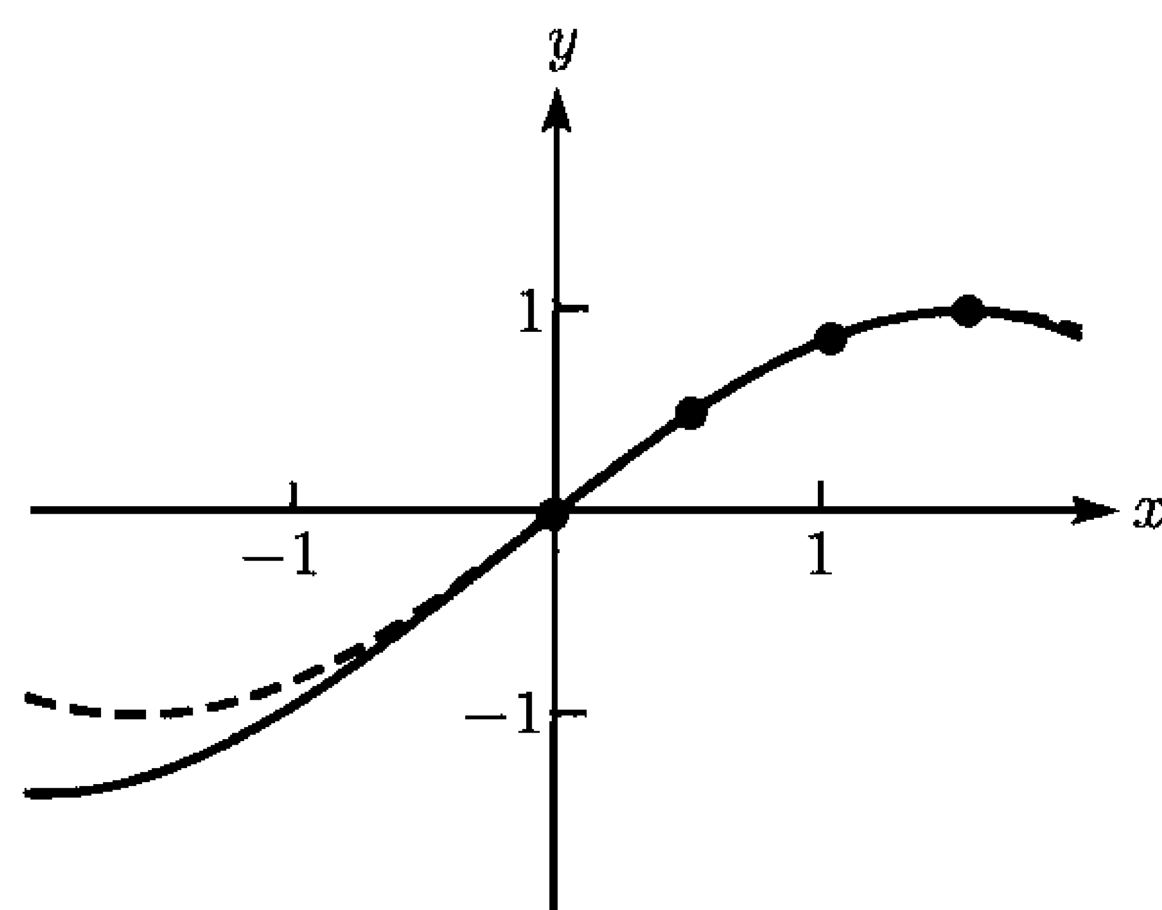


图 3-3 sin x 的 3 次插值: 沿着 $y = \sin x$ 画出插值多项式 (实曲线), 等距插值节点是 $0, \frac{\pi}{6}, \frac{2\pi}{6}$ 及 $\frac{3\pi}{6}$. 在 0 和 $\frac{\pi}{2}$ 之间插值多项式近似得非常好

在计算器上设计的正弦键的功能有多近似? 当然我们需要处理来自整个实轴的输入. 但是由于正弦函数的对称性, 我们已做了其中最佳的部分. 区间 $[0, \frac{\pi}{2}]$ 是正弦的所谓的**基本定义域**(fundamental domain), 即任何其他区间的输入都能够转变成它. 譬如从 $[\frac{\pi}{2}, \pi]$ 中给定输入 x, 因为正弦关于 $x = \frac{\pi}{2}$ 对称, 所以可以把 $\sin x$ 计算成 $\sin(\pi - x)$. 从 $[\pi, 2\pi]$ 中给定输入 x, 由于关于 $x = \pi$ 的反对称性, 所以 $\sin x = -\sin(2\pi - x)$. 最后, 因为在经过整个实轴时正弦重复它在区间 $[0, 2\pi]$ 中的性态, 所以对任何输入, 可以先约去模 2π 再进行计算. 这就导出对正弦键的直接了当的设计.

```
%Program 3.3 Building a sin calculator key, attempt #1
%Approximates sin curve with degree 3 polynomial
%     (Caution: do not use to build bridges,
%     at least until we have discussed accuracy.)
%Input:  x
%Output: approximation for sin(x)
function y=sin1(x)
%First calculate the interpolating polynomial and
%  store coefficients
b=pi*(0:3)/6;yb=sin(b);     % b holds base points
c=newtdd(b,yb,4);
%For each input x, move x to the fundamental domain and evaluate
%      the interpolating polynomial
s=1;                        % Correct the sign of sin
x1=mod(x,2*pi);
```

```
if x1>pi
  x1 = 2*pi-x1;
  s = -1;
end
if x1 > pi/2
  x1 = pi-x1;
end
y = s*nest(3,c,x1,b);
```

程序 3.3 中的大多数工作是把 x 放置到基本定义域. 然后用嵌套乘法计算三次多项式, 表 3-1 是来自程序 3.3 的一些典型的输出. 对于第一次尝试, 这个结果并不差. 误差一般在 1% 以下, 为了得到足够准确的数字来填满计算器的数字显示屏, 我们需要对插值误差懂得更多一些, 这是 3.2 节的内容.

表　3-1

x	$\sin x$	$P_3(x)$	误差
1	0.841 5	0.841 1	0.000 4
2	0.909 3	0.910 2	0.000 9
3	0.141 1	0.142 8	0.001 7
4	−0.756 8	−0.755 7	0.001 1
14	0.990 6	0.992 8	0.002 2
1 000	0.826 9	0.826 3	0.000 6

习题 3.1

1. 用 Lagrange 插值求经过下列各点的多项式.

  (a) (0, 1), (2, 3), (3, 0);

  (b) (−1, 0), (2, 1), (3, 1), (5, 2);

  (c) (0, −2), (2, 1), (4, 4);

2. 用 Newton 均差求习题 1 中各点的插值多项式, 并检验与 Lagrange 插值多项式是否一致.

3. 经过 4 个点 (−1, 3), (1, 1), (2, 3), (3, 7) 的 d 次多项式有多少个? 如果可能的话, 写出一个 (a) $d=2$; (b) $d=3$; (c) $d=6$.

4. **(a)** 求曲线经过点 (0, 0), (1, 1), (2, 2), (3, 7) 的次数小于等于 3 的多项式 $P(x)$? **(b)** 求另外两个 (任意次的) 经过这 4 个点的多项式. **(c)** 确定是否存在通过点 (0, 0), (1, 1), (2, 2), (3, 7), (4, 2) 的次数小于等于 3 的多项式 $P(x)$.

5. **(a)** 求曲线经过 4 个数据点 (−2, 8), (0, 4), (1, 2), (3, −2) 的次数小于等于 3 的多项式 $P(x)$. **(b)** 描述经过 (a) 中 4 个点的任何次数小于等于 4 的其他多项式.

6. 写出在 4 个点 (1, 1), (2, 3), (3, 3), (4, 4) 插值的次数恰为 5 的多项式.

7. 求 $P(0)$, 这里 $P(x)$ 是一个 10 次多项式, 其零点是 $x=1,\cdots,10$ 而且满足 $P(12)=44$.

8. 设 $P(x)$ 是一个 9 次多项式, 且在 $x=1$ 处取值 112, 在 $x=10$ 处取值 2, 而在 $x=2,\cdots,9$ 处都等于零. 计算 $P(0)$.

9. 给出下述的例子或者解释为什么不存在这样的例子. (a) 在 $x = 1, 2, 3, 4, 5, 6$ 处等于零, 在 $x = 7$ 处等于 10 的 6 次多项式 $L(x)$. (b) 在 $x = 1, 2, 3, 4, 5, 6$ 处等于零, 在 $x = 7$ 处等于 10, 在 $x = 8$ 处等于 70 的 6 次多项式 $L(x)$.
10. 设 $P(x)$ 是在 x=1, 2, 3, 4, 5 处取值为 10, 在 $x = 6$ 处取值 15 的 5 次多项式. 求 $P(7)$.
11. 设 P_1, P_2, P_3, P_4 是落在抛物线 $y = ax^2 + bx + c$ 上的 4 个不同的点. 经过这 4 个点的三次多项式有多少个? 对你的答案作出解释.
12. 三次多项式可能与四次多项式恰好交于 5 个点吗? 解释之.
13. 地球大气层中估计的二氧化碳平均大气浓度由表 3-2 给出 (体积浓度, 百万分率). 求这些数据的三次插值多项式, 并用它来估计 (a)1950 年 (b) 2050 年的二氧化碳浓度 (1950 年的实际浓度是 310ppm)

表 3-2

年	CO_2(ppm)
1800	280
1850	283
1900	291
2000	370

14. 当工业风扇在以下列出的温度运转时, 预期使用期限如表 3-3 所示. 通过以下方法估计在 70°C 时的使用期限. (a) 使用经过最后 3 个数据点的抛物线. (b) 使用经过全部 4 个点的三次曲线.

表 3-3

温度 (°C)	小时 (×1 000)
25	95
40	75
50	63
60	54

15. 分别用 (a) Lagrange 形式 (b) Newton 均差形式计算经过 n 个数据点的插值多项式所需要的乘法和加法次数 (用嵌套乘法计算).

计算机问题 3.1

1. 用表 3-4 中的世界人口统计数字估计 1980 年的人口, 采用 (a) 经过 1970 年和 1990 年估计值的直线, (b) 经过 1960 年、1970 年及 1990 年估计值的抛物线, (c) 经过全部 4 个数据点的三次曲线. 并与 1980 年的估计值 4 452 584 592 进行比较.

表 3-4

年	人口
1960	3 039 585 530
1970	3 707 475 887
1990	5 281 653 820
2000	6 079 603 571

2. 若输入x与y是数据点的长度相等的向量, 而输出是插值多项式的曲线, 写出程序 3.2 的 MATLAB函数形式. 用这种方式, 输入这些点能够比鼠标输入更精确. 通过复制图 3-2 来检验你的程序.
3. 取一组插值点 (x, y), 以及另一个 x_0 和输出 y_0(插值多项式在 x_0 处的值) 作为输入, 写出 MATLAB函数polyinterp.m. 文件的第一行应该是function yo=polyinterp(x,y,x0), 这里x和y是数据点的输入向量. 根据程序 3.1, 你的函数可叫作newtdd, 或者根据第 0 章可称为nest, 可以类似于程序 3.2 构造出来, 但是没有图形. 演示你的函数的功能.
4. 改变程序 3.3 中的sin1计算器键, 建立cos1键, 这是遵循相同原理的余弦键. 首先确定余弦的基本定义域.
5. (a) 用正弦和余弦的加法公式证明 $\tan(\frac{\pi}{2} - x) = \frac{1}{\tan x}$; (b) 证明 $[0, \frac{\pi}{4}]$ 可以作为 $\tan x$ 的基本定义域; (c) 按照程序 3.3 的原理, 在这个基本定义域上采用三次插值多项式来设计正切键; (d) 根据经验计算正切键在 $[0, \frac{\pi}{4}]$ 中的最大误差.

3.2 插值误差

正弦计算器键的精度依赖于图 3-3 中的近似, 它有多接近? 我们曾给出过表明这种情形的表格. 对于某些示例而言, 前面两个数字是完全可靠的, 但是后面的数字并不总是正确. 在这一节, 我们将研究度量这种误差的方法以及确定如何使它更小.

3.2.1 插值误差公式

假设我们从函数 $y = f(x)$ 出发, 并且从它取出数据点来构造插值多项式 $P(x)$, 就像我们在例 3.7 中对 $f(x) = \sin x$ 所做的那样. 在 x 处的**插值误差**(interpolation error) 是 $f(x) - P(x)$, 即在 x 处计算提供数据点的原函数与插值多项式的差. 插值误差是图 3-3 中两条曲线之间的垂直距离. 下面的定理给出了插值误差的公式, 通常它不可能准确地计算, 但是至少可以导出误差界.

定理 3.3 假设 $P(x)$ 是拟合 n 个点 $(x_1, y_1), \cdots, (x_n, y_n)$ 的次数小于等于 $n-1$ 的插值多项式. 插值误差是

$$f(x) - P(x) = \frac{(x - x_1)(x - x_2)\cdots(x - x_n)}{n!} f^{(n)}(c), \tag{3.6}$$

这里 c 落在 $x, x_1, \cdots, x_n$ 中的最小值与最大值之间.

定理 3.3 的证明见 3.2.2 节. 可以用这个定理去估计我们在例 3.7 中建立的正弦键的精度. 由等式 (3.6) 得到

$$\sin x - P(x) = \frac{(x - 0)(x - \frac{\pi}{6})(x - \frac{\pi}{3})(x - \frac{\pi}{2})}{4!} f''''(c),$$

这里 $0 < c < \frac{\pi}{2}$. 4 阶导数 $f''''(c) = \sin c$ 在这个范围内从 0 到 1 变化. 即使是最坏的情形, $|\sin c|$ 也不会大于 1, 因此我们可以确定插值误差的上界.

$$|\sin x - P(x)| \leqslant \frac{|(x-0)(x-\frac{\pi}{6})(x-\frac{\pi}{3})(x-\frac{\pi}{2})|}{24}|1|.$$

在 $x=1$ 处, 最坏情形的误差是

$$|\sin 1 - P(1)| \leqslant \frac{|(1-0)(1-\frac{\pi}{6})(1-\frac{\pi}{3})(1-\frac{\pi}{2})|}{24}|1| \approx 0.000\,534\,8. \tag{3.7}$$

因为我们用了关于 4 阶导数的最坏情形的界, 所以这是误差的上界. 注意在 $x=1$ 处的实际误差是 0.000 4, 它是在由 (3.7) 给出的误差界之内的. 在这种形式的插值误差公式的基础上, 我们可以得出一些结论. 当 x 较接近 x_i 的区间的中点时, 我们预计比它靠近其中一个端点时的误差更小, 这是因为在乘积中将出现更小的项. 例如, 我们把前面的误差界与 $x=0.2$ 的情形 (它靠近数据点范围的左端点) 进行比较, 此时误差公式是

$$|\sin 0.2 - P(0.2)| \leqslant \frac{|(0.2-0)(0.2-\frac{\pi}{6})(0.2-\frac{\pi}{3})(0.2-\frac{\pi}{2})|}{24}|1| \approx 0.003\,13,$$

大约是原先的 6 倍, 相应地, 实际误差也较大, 具体为

$$|\sin 0.2 - P(0.2)| = |0.198\,67 - 0.200\,56| = 0.001\,89.$$

例 3.8 求 $f(x)=\mathrm{e}^x$ 与在点 $-1, -0.5, 0, 0.5, 1$ 处插值的多项式在 $x=0.25$ 及 $x=0.75$ 处的差的上界.

构造如图 3-4 所示的插值多项式, 它对于求这种上界并不是必需的. 插值误差公式 (3.6) 给出

$$f(x) - P_4(x) = \frac{(x+1)(x+\frac{1}{2})x(x-\frac{1}{2})(x-1)}{5!}f^{(5)}(c),$$

这里 $-1<c<1$, 5 阶导数 $f^{(5)}(c)=\mathrm{e}^c$. 因为 e^x 关于 x 递增, 它的最大值在区间的右端点, 所以在 $[-1,1]$ 上, $|f^{(5)}| \leqslant \mathrm{e}^1$ 对于 $-1 \leqslant x \leqslant 1$, 误差公式变为

$$|\mathrm{e}^x - P_4(x)| \leqslant \frac{(x+1)(x+\frac{1}{2})x(x-\frac{1}{2})(x-1)}{5!}\mathrm{e}.$$

在 $x=0.25$ 处, 插值误差有上界

$$|\mathrm{e}^{0.25} - P_4(0.25)| \leqslant \frac{(1.25)(0.75)(0.25)(-0.25)(-0.75)}{120}\mathrm{e} \approx 0.000\,995.$$

在 $x=0.75$ 处, 插值误差可能比较大:

$$|e^{0.75} - P_4(0.75)| \leqslant \frac{(1.75)(1.25)(0.75)(0.25)(0.25)}{120}e \approx 0.002\ 323.$$

再次注意, 在靠近插值区间中央的插值误差将变得较小. ◀

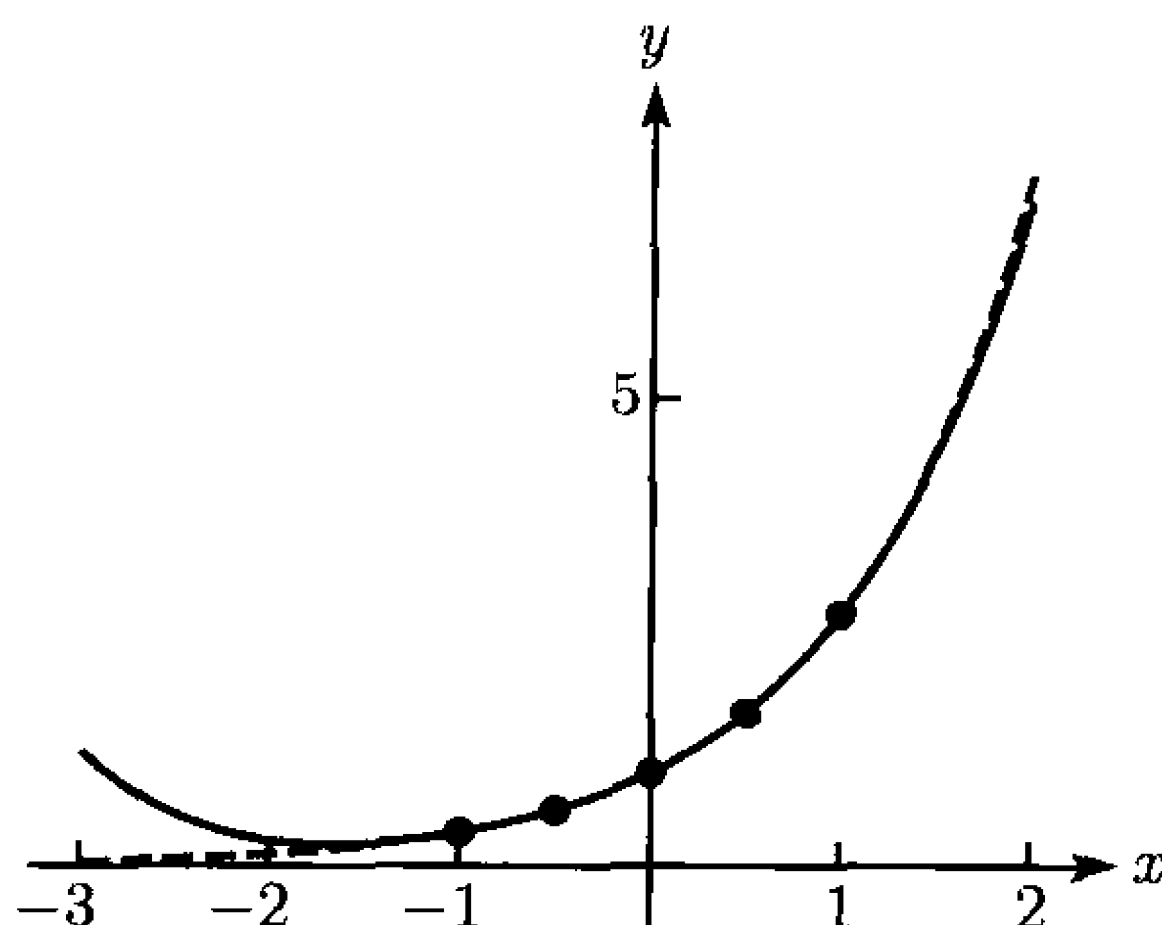


图 3-4 对 $f(x) = e^x$ 进行近似的插值多项式等距基点 $-1, -0.5, 0, 0.5, 1$, 实曲线是插值多项式

3.2.2 Newton 形式和误差公式的证明

在这一节中, 我们来解释早先使用过的两个重要事实背后的原理. 首先, 我们建立插值多项式的 Newton 均差形式, 然后我们证明插值误差公式.

回忆我们至今所知道的事情. 如果 $x_1, \cdots, x_n$ 是实轴上 n 个不同的点, $y_1, \cdots, y_n$ 是任意的, 我们知道对于这些点恰好存在一个 (次数最多为 $n-1$) 插值多项式 $P_{n-1}(x)$. 我们也知道 Lagrange 插值公式给出了这种多项式.

只有一件事情我们不知道, 那就是 Newton 均差公式是否也给我们一个插值多项式. 一旦在定理 3.7 中证明它是对的, 我们就将知道它必须与 Lagrange 形式一致. 我们从求 $n-1$ 次的 P_{n-1} 的系数公式开始, 然后再专门研究 Newton 形式.

考虑另外两个插值多项式: 插值于 $x_1, \cdots, x_{n-1}$ 的 P_{n-2}, 以及插值于 $x_2, \cdots, x_n$ 的 Q_{n-2}. 它们之间成立以下关系:

引理 3.4 $(P_{n-1}(x) - P_{n-2}(x))(x_n - x_1) = (x - x_1)(Q_{n-2}(x) - P_{n-2}(x))$.

证 因为两边的差是一个最多 $n-1$ 次的多项式, 所以只要证明在 n 个不同的点两边相等就足够了. 于是根据代数学基本定理, 这个差必定恒等于零多项式. 我们选取的 n 个点是 n 个插值点. 在 $x = x_1$ 处, 左边是零, 这是因为 P_{n-1} 和 P_{n-2} 都在点 (x_1, y_1) 插值. 右边也是零. 在 $x = x_2, \cdots, x_{n-1}$ 处两边都是零, 这是因为全部 3 个多项式 $P_{n-1}, P_{n-2}, Q_{n-2}$ 都在这些点插值. 对于 $x = x_n$, 两边是恒等的, 这是因为 $P_{n-1}(x_n) = Q_{n-2}(x_n)$.

定义 3.5 我们把插值于 $x_1, \cdots, x_n$ 且次数小于等于 $n-1$ 的唯一多项式的 $n-1$ 次项系数记为 $f[x_1 \cdots x_n]$.

用这个定义, 引理 3.4 中左边的 $n-1$ 次系数是 $f[x_1\cdots x_n](x_n-x_1)$, 而右边的 $n-1$ 次系数是 $f[x_2\cdots x_n]-f[x_1\cdots x_{n-1}]$. 因为它们必须相等, 所以我们已经证明了下面的引理.

引理 3.6 如果 $f[x_1\cdots x_n]$ 表示插值于 $x_1,\cdots,x_n$ 且次数小于等于 $n-1$ 的多项式的 $n-1$ 次项系数, 那么

$$f[x_1\cdots x_n]=\frac{f[x_2\cdots x_n]-f[x_1\cdots x_{n-1}]}{x_n-x_1}. \tag{3.8}$$

现在我们知道如何求 $n-1$ 次系数. 利用 (3.8) 的递归性质, 通过应用 Newton 均差三角形, 我们就能够计算它.

因此, 我们讨论专门针对 Newton 形式:

$$P_{n-1}(x)=a_1+a_2(x-x_1)+a_3(x-x_1)(x-x_2)+\cdots+a_n(x-x_1)\cdots(x-x_{n-1}). \tag{3.9}$$

首先, 是否永远可能把插值多项式写成这种形式? 通过求出系数 a_i 用数据点 y_i 来表示, 我们证明这是可能的. 例如, 把 x_1 代入 Newton 形式 (3.9) 得到

$$a_1=P_n(x_1)=y_1. \tag{3.10}$$

代入 x_2 得到

$$a_1+a_2(x_2-x_1)=P_{n-1}(x_2)=y_2, \tag{3.11}$$

所以我们也能解出 a_2, 这是因为根据 (3.10), a_1 是已知的, 并且 a_2 的系数非零. 代入 x_3 我们看到同样的结果:

$$a_1+a_2(x_3-x_1)+a_3(x_3-x_1)(x_3-x_2)=P_{n-1}(x_3)=y_3. \tag{3.12}$$

一般地,

$$\begin{aligned}&a_1+a_2(x_m-x_1)+a_3(x_m-x_1)(x_m-x_2)\\&+\cdots+a_m(x_m-x_1)\cdots(x_m-x_{m-1})\\=&P_{n-1}(x_m)=y_m.\end{aligned} \tag{3.13}$$

重要的事实是前面所有的 a_i 都是已知的, 而且等式中 a_m 的系数非零, 这是因为所有的点 x_i 都是不同的.

现在我们知道, Newton 形式永远是存在的, 我们来说明如何计算 a_m. 当然, 我们从引理 3.6 已经知道 $n-1$ 次系数 $a_n=f[x_1\cdots x_n]$. 对于写成 Newton 形式的多项式, 用 $(p_{n-1})_{m-1}$ 表示由 P_{n-1} 的 Newton 形式的前 m 项组成的次数小于等于 $m-1$ 的多项式. 注意 $(p_{n-1})_{m-1}$ 插值于 $(x_1,y_1),\cdots,(x_m,y_m)$, 这是因为 P_{n-1} 确实是这样, 而且在这些点处较高项全是零. 根据插值多项式的唯一性,

$(P_{n-1})_{m-1}$ 是次数小于等于 $m-1$ 且在 $x_1, \cdots, x_m$ 处插值的多项式, 所以根据引理 3.6,$a_m = f[x_1 \cdots x_m]$. 这样我们已经证明了以下定理:

定理 3.7 给定 n 个点 $(x_1, y_1), \cdots, (x_n, y_n)$ 使得 x_i 各不相同, 写成 Newton 形式 (3.9) 的多项式 $P_{n-1}(x)$ 具有

$$a_m = f[x_1 \cdots x_m], \quad 1 \leqslant m \leqslant n$$

给出的系数. 换言之, Newton 均差公式给出了经过 n 个点且次数小于等于 $n-1$ 的唯一的插值多项式.

下面证明插值误差定理 3.3. 考虑另加一点 x 到这组插值点中. 新的插值多项式应该是

$$P_n(t) = P_{n-1}(t) + f[x_1 \cdots x_n x](t - x_1) \cdots (t - x_n).$$

在这个额外的点 x 处计算, $P_n(x) = f(x)$, 所以

$$f(x) = P_{n-1}(x) + f[x_1 \cdots x_n x](x - x_1) \cdots (x - x_n). \tag{3.14}$$

这个公式对所有的 x 都正确. 现在定义

$$h(t) = f(t) - P_{n-1}(t) - f[x_1 \cdots x_n x](t - x_1) \cdots (t - x_n).$$

注意, 根据 (3.14), $h(x) = 0$, $0 = h(x_1) = \cdots = h(x_n)$, 这是因为 P_{n-1} 在这些点对 f 进行插值. 在这 $n+1$ 个点 $x, x_1, \cdots, x_n$ 的每一对相邻点之间, 根据 Rolle 定理 (见第 0 章) 必定存在一个新的点, 在这点上 $h' = 0$. 一共有 n 个这样的点. 在这些点的每一对点之间, 必定存在一个新的点, 在这点上 $h'' = 0$, 一共有 $n-1$ 个这样的点. 以这种方式进行下去, 必定存在一个点 c 使得 $h^{(n)}(c) = 0$, 这里的 c 落在 $x, x_1, \cdots, x_n$ 中的最小值和最大值之间. 注意到

$$h^{(n)}(t) = f^{(n)}(t) - n! f[x_1 \cdots x_n x],$$

这是因为多项式 $P_{n-1}(x)$ 的 n 阶导数等于 0. 代入 c 就得到

$$f[x_1 \cdots x_n x] = \frac{f^{(n)}(c)}{n!},$$

使用 (3.14), 就能导出

$$f(x) = P_{n-1}(x) + \frac{f^{(n)}(c)}{n!}(x - x_1) \cdots (x - x_n).$$

3.2.3 Runge 现象

如定理 3.2 所示, 多项式能够拟合任何一组数据点. 但是某些多项式的曲线偏好于某种特定形状. 通过使用程序 3.2, 你可能会较好地了解这一点. 尝试使函数在

等距点 $x = -3, -2.5, -2, -1.5, \cdots, 2.5, 3$ 处等于 0, $x = 0$ 除外, 在 $x = 0$ 处我们置函数值等于 1. 除了在 $x = 0$ 处有一个三角形的 "隆起" 之外, 数据点沿着 x 轴是平坦的, 如图 3-5 所示.

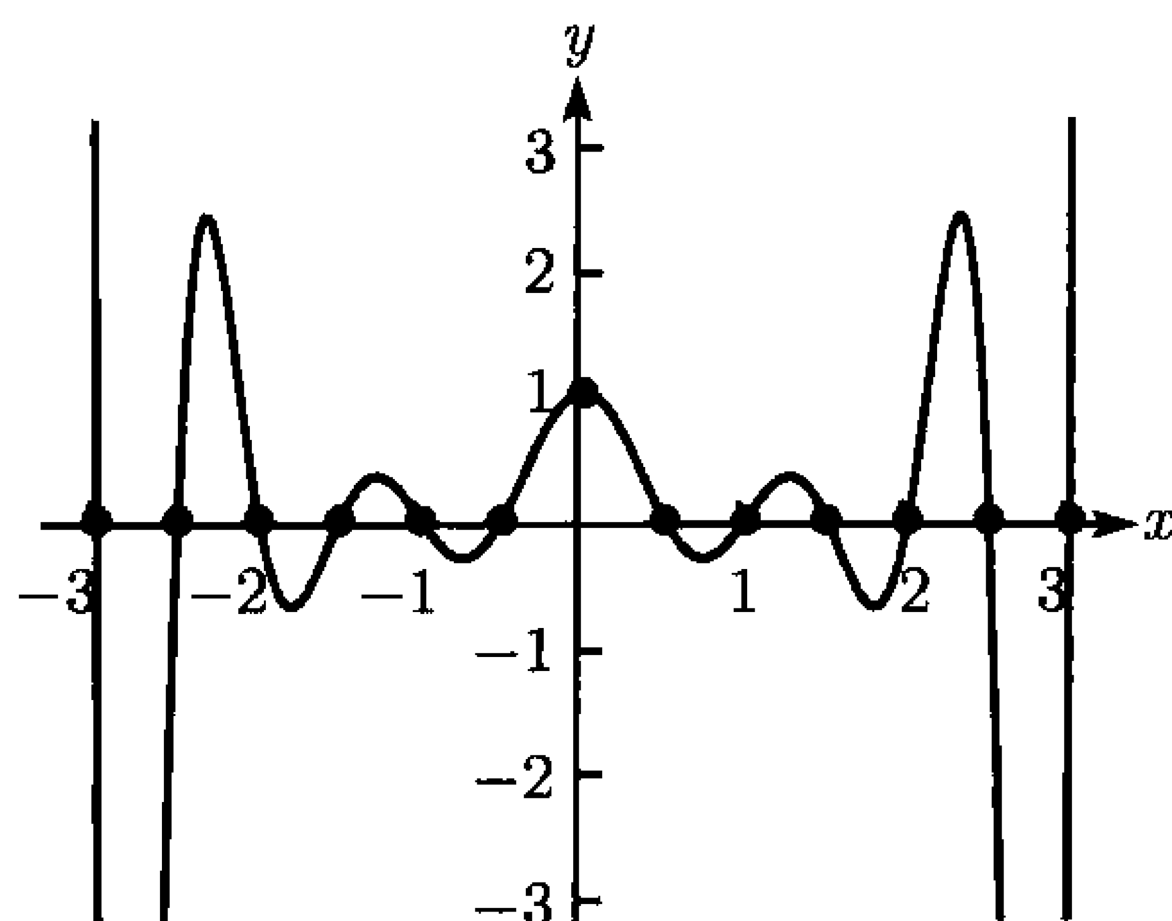


图 3-5 三角形隆起函数的插值：插值多项式比输入数据点摆动更甚

经过这样设置的点的多项式与数据点不同, 其值拒绝停留在 0 与 1 之间. 这就是所谓的 **Runge 现象**. 它通常用来描述与在等距点插值的高次多项式相关的极端的 "多项式摆动" 现象.

例 3.9 把函数 $f(x) = \frac{1}{1+12x^2}$ 在 $[-1, 1]$ 中等距的点上插值.

这称为 **Runge 例子**. 该函数具有与图 3-5 中三角形隆起相同的一般形状. 图 3-6 展示了插值的结果, 即 Runge 现象特征的效应：在插值区间端点附近的多项式摆动. ◀

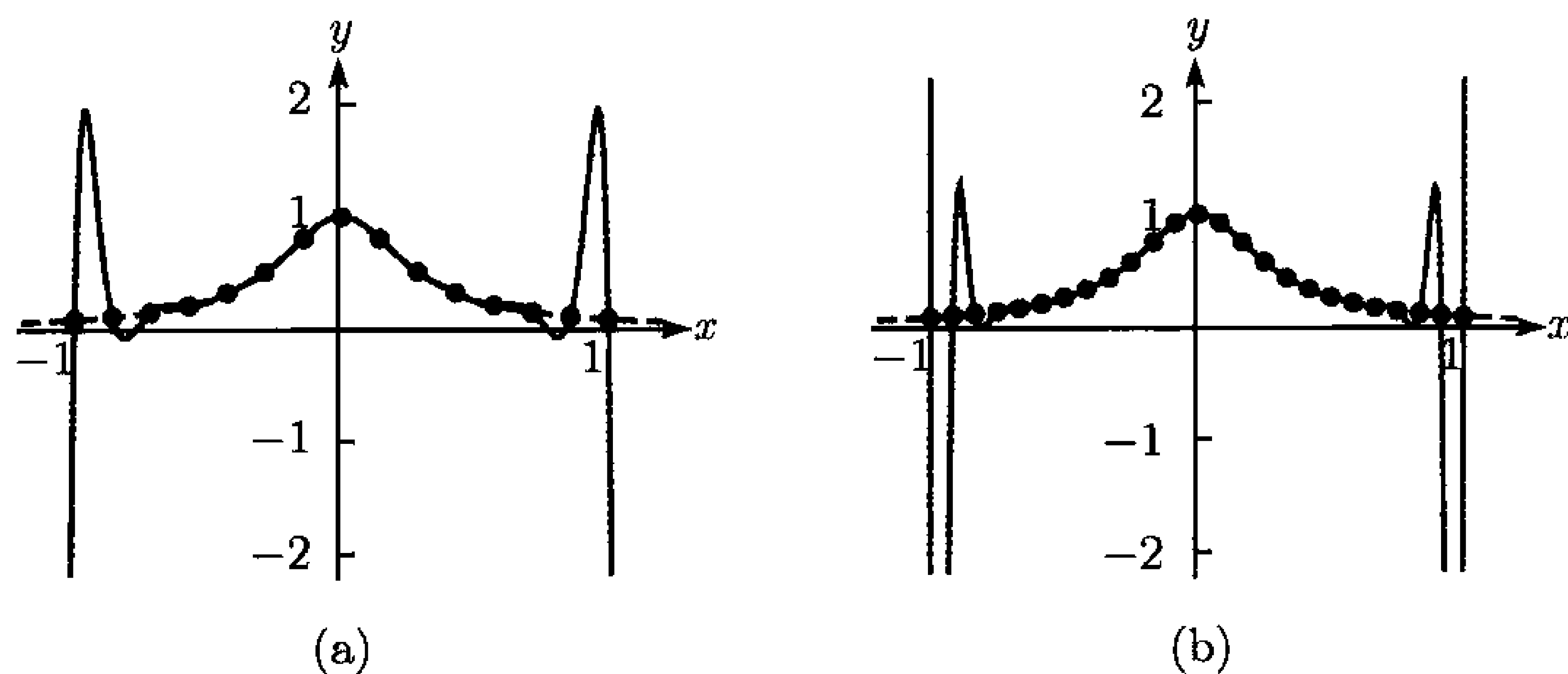


图 3-6 Runge 例子. 例 3.9 中的 Runge 函数在等距基点处的多项式插值类似于图 3-5, 它在区间的端点附近变化剧烈：(a) 15 个基点, (b) 25 个基点

如我们已经看到的, 带有 Runge 现象的例子有这样一种特征：它们在靠近数据点区间的外面时有大的误差. 对这个问题的解决是靠直觉的：把一些插值点朝区间外部移动, 在那里产生数据的函数能更好地拟合. 3.3 节关于 Chebyshev 插值的讨论将介绍如何完成这一点.

习题 3.2

1. (a) 求经过点 (0, 0),$(\frac{\pi}{2},1)$, $(\pi,0)$ 的插值多项式 $P_2(x)$. (b) 计算 $\sin(\frac{\pi}{4})$ 的近似 $P_2(\frac{\pi}{4})$. (c) 用定理 3.3 对 (b) 中的近似给出误差界. (d) 用计算器或者 MATLAB, 比较实际误差和你的误差界.
2. (a) 给定数据点 (1, 0), (2, ln 2), (4, ln 4), 求二次插值多项式. (b) 用 (a) 的结果去近似 ln 3. (c) 用定理 3.3 对 (b) 中的近似给出误差界. (d) 比较实际误差和你的误差界.
3. 设多项式 $f(x)=\mathrm{e}^{-2x}$ 在 10 个等距的点 $x=0, \frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \cdots, \frac{8}{9}, 1$ 的插值函数为 $P_9(x)$. (a) 求误差 $|f(\frac{1}{2})-P_9(\frac{1}{2})|$ 的上界; (b) 如果用 $P_9(\frac{1}{2})$ 来近似 e^{-1}, 你能保证多少位小数是正确的?
4. 考虑插值节点 $x=0, 2, 4, 6, 8, 10$, 求 $f(x)=\frac{1}{x+5}$ 的插值多项式. 在 (a)$x=1$ 及 (b)$x=5$ 处求插值误差的上界.
5. 采用数据点 $(x_i, f(x_i))$, 这里 $x_1=0.1, x_2=0.2, x_3=0.3, x_4=0.4, x_5=0.5, x_6=0.6$, 假设已经用 5 次插值多项式来逼近函数 $f(x)$. 对于 $x=0.35$ 或 $x=0.55$, 你预计插值误差 $|f(x)-P(x)|$ 比较小吗? 量化你的回答.
6. 假设多项式 $P_5(x)$ 在 6 个数据点 $(x_i, f(x_i))$(其 x 坐标分别是 $x_1=0, x_2=0.2, x_3=0.4, x_4=0.6, x_5=0.8, x_6=1$) 对函数 $f(x)$ 插值. 假设在 $x=0.3$ 处的插值误差是 $|f(0.3)-P_5(0.3)|=0.01$. 如果再加上两个插值点 $(x_6,y_6)=(0.1,f(0.1))$ 及 $(x_7,y_7)=(0.5,f(0.5))$, 估计新的插值误差 $|f(0.3)-P_7(0.3)|$ 将是多少. 为了得到这个估计, 你已作了什么假设?

计算机问题 3.2

1. (a) 对数据 (0.6, 1.433 329), (0.7, 1.632 316), (0.8, 1.896 481), (0.9, 2.247 908), (1.0, 2.718 282) 用均差方法求 4 次插值多项式 $P_4(x)$; (b) 计算 $P_4(0.82)$ 及 $P_4(0.98)$; (c) 前面的数据来自函数 $f(x)=\mathrm{e}^{x^2}$, 用插值误差公式求在 $x=0.82$ 及 $x=0.98$ 处的误差上界, 并且把这个界与实际误差进行比较; (d) 在区间 [0.5, 1] 及 [0, 2] 上画出实际插值误差 $P_4(x)-\mathrm{e}^{x^2}$.
2. 在区间 $[-2\pi, 2\pi]$ 上画出程序 3.3 中 sin 1 键的插值误差.
3. 世界石油产量以每天百万桶计, 如表 3-5 所示. 确定并画出经过这些数据的 9 次多项式. 用它估计 2010 年的石油产量. 在这个例子里, Runge 现象会出现吗? 你以为插值多项式是数据的好模型吗? 请解释之.

表 3-5

年	桶/天 ($\times 10^6$)	年	桶/天 ($\times 10^6$)
1994	67.052	1999	72.063
1995	68.008	2000	74.669
1996	69.803	2001	74.487
1997	72.024	2002	74.065
1998	73.400	2003	76.777

4. 用经过计算机问题 3 中前 4 个数据点的三次多项式估计 1998 年世界石油产量. Runge 现象会出现吗?

3.3 Chebyshev 插值

通常插值选取的基点 x_i 是等距的. 在许多情形下, 被插值的数据仅仅适用于这种形式, 例如, 当数据是由分成常数时间区间的仪表读数所组成时. 在其他情形(譬如, 正弦键) 我们可以自由选取认为合适的基点. 于是基点间隔的选取对插值误差有重大的影响. Chebyshev 插值涉及间隔点的一种特殊的最优选取方式.

3.3.1 Chebyshev 定理

Chebyshev 插值用于改善在插值区间的插值误差

$$\frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{n!}f^{(n)}(c)$$

的最大值的控制. 现在固定此区间是 $[-1, 1]$.

插值误差公式中的分子

$$(x-x_1)(x-x_2)\cdots(x-x_n) \tag{3.15}$$

本身是一个 x 的 n 次多项式并且在 $[-1, 1]$ 上存在最大值, 在 $[-1, 1]$ 中是否有可能求出特定的 $x_1,\cdots,x_n$ 使得 (3.15) 式的最大值尽可能地小？这就叫做插值的极小极大值 (minimax) 问题.

例如, 图 3-7a 显示了当 $x_1,\cdots,x_9$ 是等距时 9 次多项式 (3.15) 的图形. 在靠近区间 $[-1, 1]$ 的端点时这个多项式将变大的趋势是 Runge 现象的一种体现. 图 3-7b 展示了同样的多项式 (3.15), 但是其中的点 $x_1,\cdots,x_n$ 已经用这样的方式选择, 它使多项式的大小在整个 $[-1,1]$ 内相等. 这些点已经按照即将提出的定理 3.8 进行了选取.

事实上, 这种准确的定位, 其中选取的基点 x_i 是 $\cos\frac{\pi}{18},\cos\frac{3\pi}{18},\cdots,\cos\frac{17\pi}{18}$, 使得 (3.15) 的最大绝对值等于 $\frac{1}{256}$, 这是 9 个点在 $[-1, 1]$ 中可能的最小值. 这样的定位, 归因于 Chebyshev, 概述在以下定理中:

定理 3.8 使 $\max\limits_{-1\leqslant x\leqslant 1}|(x-x_1)\cdots(x-x_n)|$ 的值尽可能小的实数 $-1\leqslant x_1,\cdots,x_n\leqslant 1$ 的选取是 $x_i=\cos\frac{(2i-1)\pi}{2n}, i=1,\cdots,n$, 而且最小值是 $\frac{1}{2^{n-1}}$. 事实上, 最小值是通过

$$(x-x_1)\cdots(x-x_n)=\frac{1}{2^{n-1}}T_n(x)$$

达到的, 这里 $T_n(x)$ 是 n 次 Chebyshev 多项式.

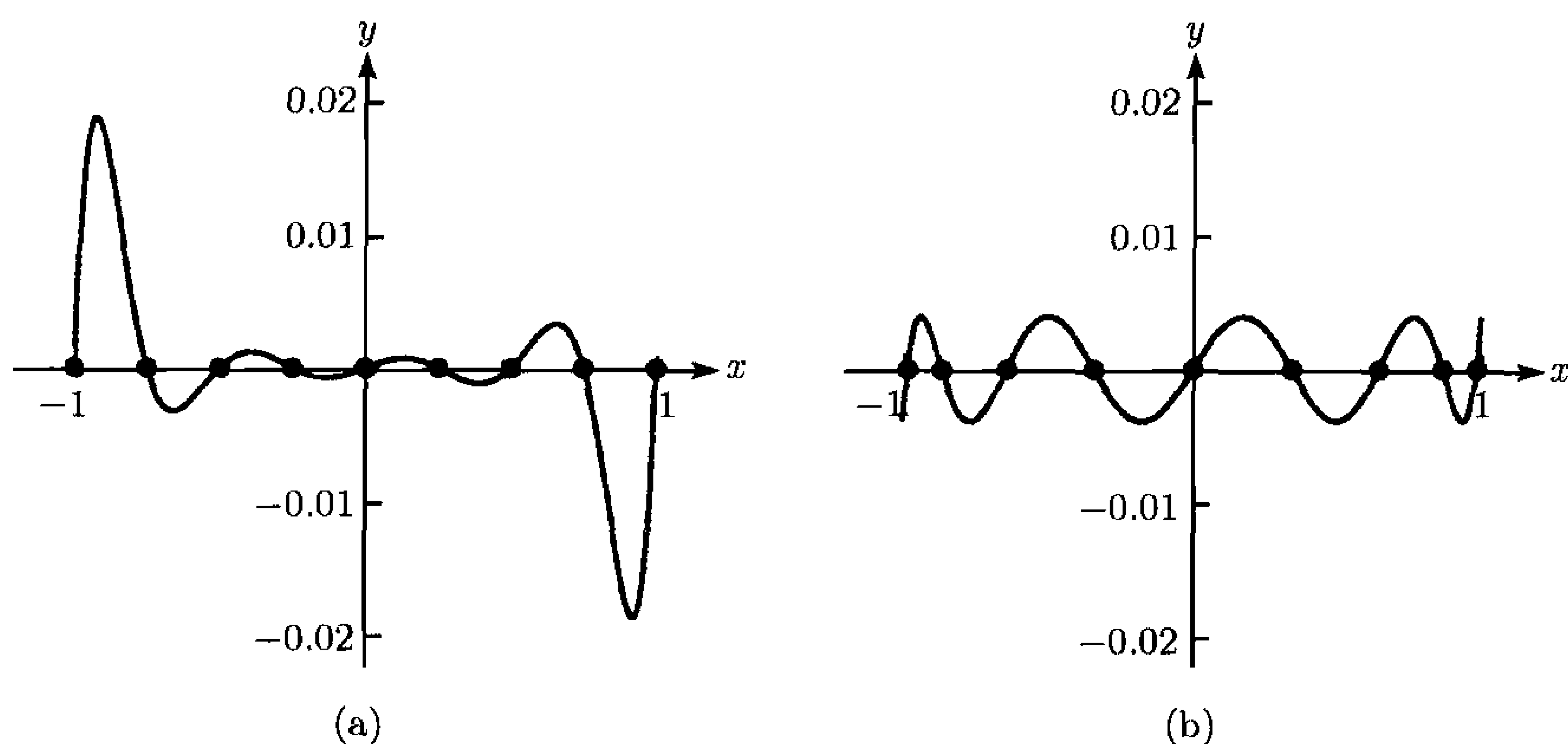


图 3-7 插值误差公式的一部分.$(x-x_1)\cdots(x-x_9)$ 的图形: (a) 9 个等距的基点 x_i, (b) 9 个 Chebyshev 根 x_i

在建立 Chebyshev 多项式的一些性质之后, 我们给出这个定理的证明. 从这个定理, 我们推知, 如果 $[-1, 1]$ 中的 n 个插值基点选取为 n 次 Chebyshev 插值多项式 $T_n(x)$ 的根, 那么就能使插值误差最小化. 这些根是

$$x_i = \cos\frac{\text{odd}\ \pi}{2n} \tag{3.16}$$

这里 "odd" 表示从 1 到 $2n-1$ 中的奇数. 于是我们保证了 (3.15) 的绝对值对于 $[-1, 1]$ 中的所有 x 是小于 $\frac{1}{2^{n-1}}$ 的.

选取 Chebyshev 根作为插值基点使得插值误差在整个区间 $[-1, 1]$ 中尽可能均匀地分布. 我们将把采用 Chebyshev 根作为基点的插值多项式称为 **Chebyshev 插值多项式.**

例 3.10 求 $f(x) = \mathrm{e}^x$ 和 4 次 Chebyshev 插值多项式在 $[-1, 1]$ 上的差在最坏情形下的误差界. 插值误差公式 (3.6) 给出

$$f(x) - P_4(x) = \frac{(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{5!}f^{(5)}(c),$$

这里

$$x_1 = \cos\frac{\pi}{10},\quad x_2 = \cos\frac{3\pi}{10},\quad x_3 = \cos\frac{5\pi}{10},\quad x_4 = \cos\frac{7\pi}{10},\quad x_5 = \cos\frac{9\pi}{10}$$

是 Chebyshev 根, 其中 $-1 < c < 1$. 根据 Chebyshev 定理 3.8, 对于 $-1 < x < 1$,

$$|(x-x_1)\cdots(x-x_5)| \leqslant \frac{1}{2^4}.$$

而且, $|f^{(5)}| \leqslant \mathrm{e}^1$ 在 $[-1, 1]$ 上成立. 插值误差是

$$|\mathrm{e}^x - P_4(x)| \leqslant \frac{\mathrm{e}}{2^4 5!} \approx 0.001\ 42,$$

对区间 $[-1, 1]$ 上的所有 x 都成立.

把这个结果与例 3.8 进行比较. Chebyshev 插值在整个区间上的误差界仅仅比用等距插值时靠近区间中央的点的误差界稍微大一些. 在靠近区间的端点处, Chebyshev 误差要小得多. ◀

回到 Runge 例题 3.9, 我们可以按照 Chebyshev 的想法通过选取插值点来消除 Runge 现象. 图 3-8 表明在整个区间 $[-1, 1]$ 插值误差都已经变小.

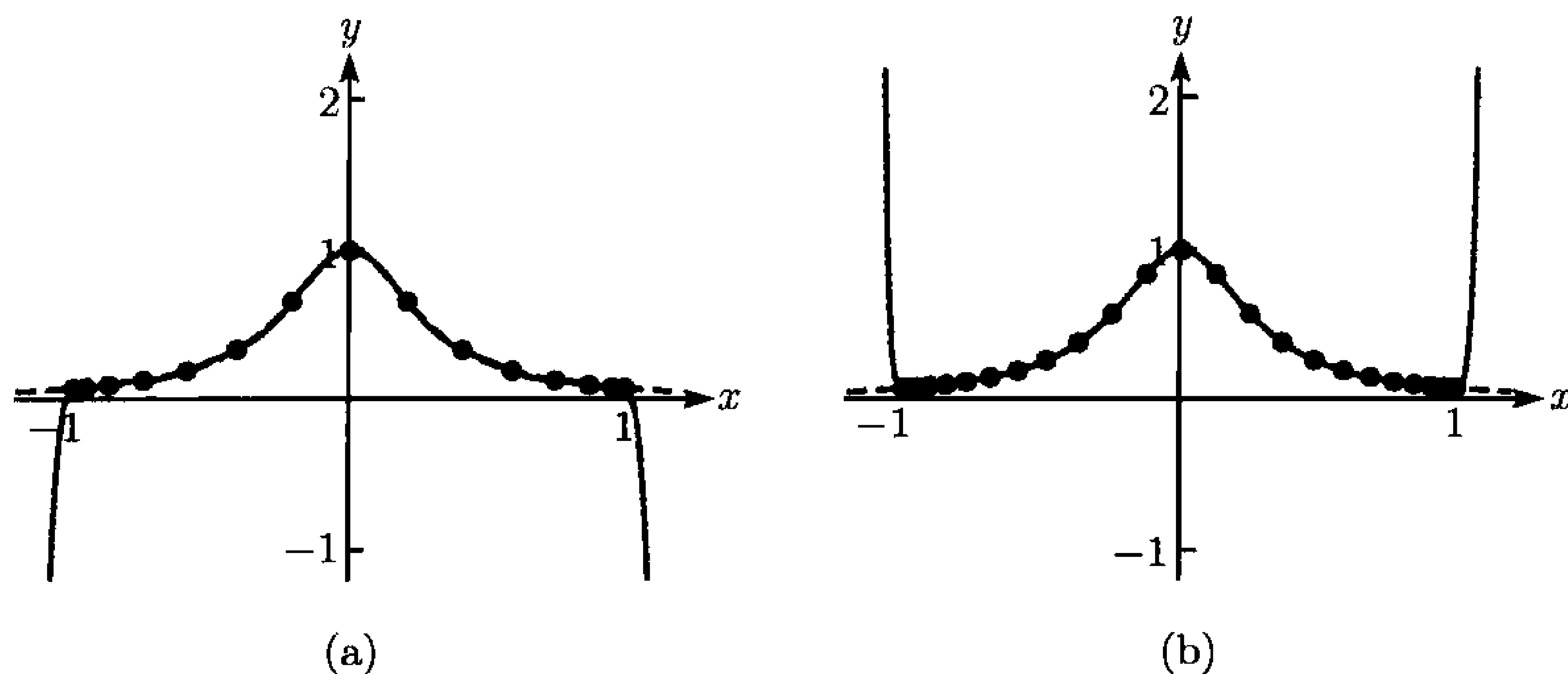


图 3-8 用 Chebyshev 节点的 Runge 例子的插值：分别对 (a)15 个点和 (b)25 个点绘出了 Runge 函数 $f(x) = \frac{1}{1+12x^2}$ 以及它的 Chebyshev 插值多项式. 在这种分辨率下, $[-1, 1]$ 上的误差可忽略不计. 至少在 -1 和 1 之间已经消除了图 3-6 中的多项式摆动

3.3.2 Chebyshev 多项式

用 $T_n(x) = \cos(n \arccos x)$ 定义 n 次 Chebyshev 多项式. 尽管它看起来不像, 但对每一个 n, 它确是变量 x 的多项式. 例如, 对 $n = 0$, 它给出零次多项式 1. 而对 $n = 1$, 我们得到 $T_1(x) = \cos(\arccos x) = x$. 对 $n = 2$, 回忆起余弦的加法公式 $\cos(a + b) = \cos a \cos b - \sin a \sin b$. 令 $y = \arccos x$, 所以 $\cos y = x$, 于是 $T_2(x) = \cos 2y = \cos^2 y - \sin^2 y = 2\cos^2 y - 1 = 2x^2 - 1$, 这是一个二次多项式. 一般地, 注意到

$$\begin{aligned} T_{n+1}(x) &= \cos(n+1)y = \cos(ny + y) = \cos ny \cos y - \sin ny \sin y, \\ T_{n+1}(x) &= \cos(n-1)y = \cos(ny - y) = \cos ny \cos y - \sin ny \sin(-y). \end{aligned} \tag{3.17}$$

因为 $\sin(-y) = -\sin y$, 所以我们可以把上面的等式相加得到

$$T_{n+1}(x) + T_{n-1}(x) = 2\cos ny \cos y = 2xT_n(x). \tag{3.18}$$

结果得到关系

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \tag{3.19}$$

称之为 Chebyshev 多项式的**递推关系**(recursion relation). 从 (3.19) 得到以下一些事实:

事实 1 T_n 是多项式. 我们直接证明了 T_0, T_1 及 T_2. 因为 T_3 是 T_1 和 T_2 的多项式组合, 所以 T_3 也是多项式. 同样的论证适用于所有的 T_n. 前面的一些 Chebyshev 多项式 (见图 3-9) 是

$$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_2(x) = 2x^2 - 1, \qquad T_3(x) = 4x^3 - 3x.$$

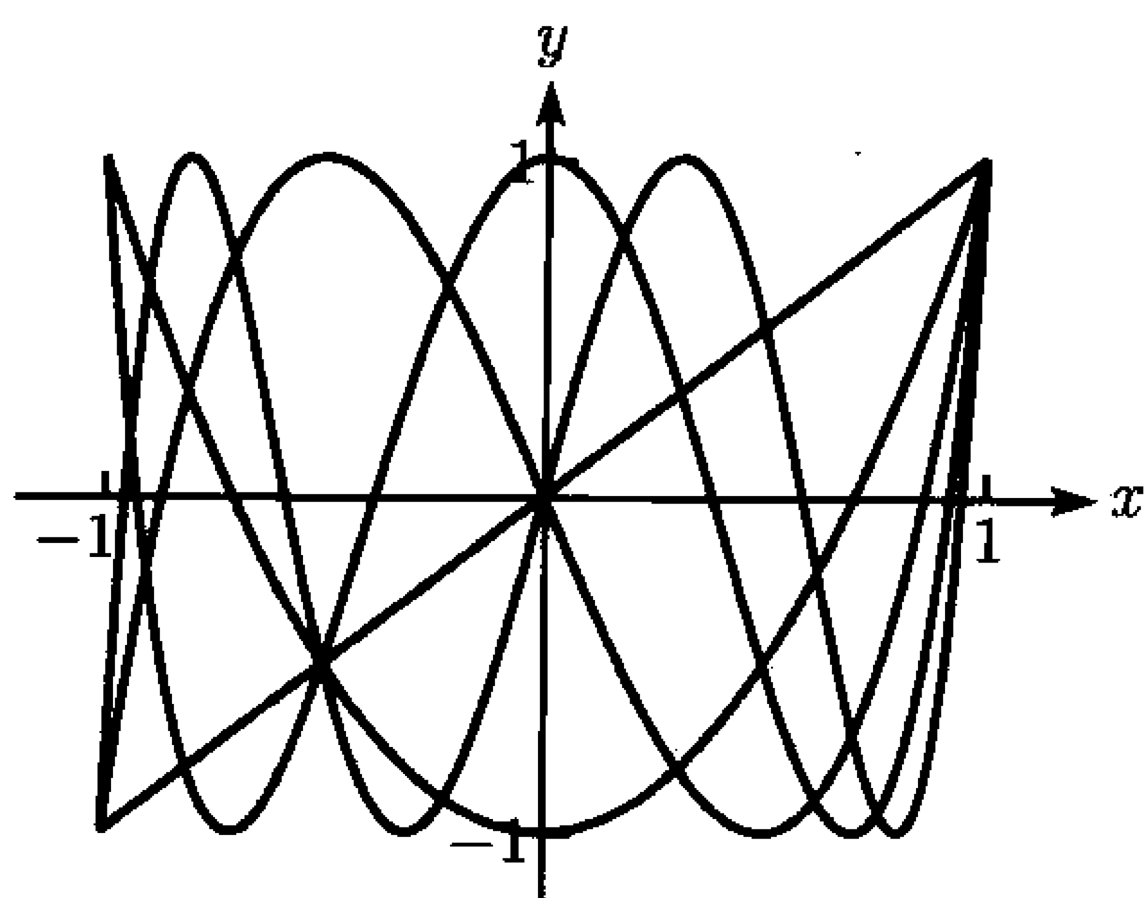


图 3-9 一次到五次 Chebyshev 多项式的图形. 注意, T_n (1) =1 以及在 [−1, 1] 内由 $T_n(x)$ 取的最大绝对值是 1

事实 2 $\deg(T_n) = n$ 并且首项系数是 2^{n-1}. 对于 $n = 1$ 及 $n = 2$, 这是很显然的. 而且递推关系把这个事实推广到所有的 n.

事实 3 $T_n(1) = 1$ 及 $T_n(-1) = (-1)^n$. 对于 $n = 1$ 及 $n = 2$, 这两点是显然的. 一般地,

$$T_{n+1}(1) = 2(1)T_n(1) - T_{n-1}(1) = 2(1) - 1 = 1$$

$$\begin{aligned} T_{n+1}(-1) &= 2(-1)T_n(-1) - T_{n-1}(-1) = -2(-1)^n - (-1)^{n-1} \\ &= (-1)^{n-1}(2-1) = (-1)^{n-1} = (-1)^{n+1}. \end{aligned}$$

事实 4 对 $-1 \leqslant x \leqslant 1$, $T_n(x)$ 的最大绝对值是 1. 这一点可从下面这个事实得到, 即对某些 y, $T_n(x) = \cos y$.

事实 5 $T_n(x)$ 的全部零点位于 −1 和 1 之间. 见图 3-10. 事实上, 这些零点是 $0=\cos(n \arccos x)$ 的解. 因为 $\cos y = 0$ 当且仅当 $y =$ 奇数 $\times(\pi/2)$, 我们得到

$$n \arccos x = \text{奇数} \times \pi/2,$$

$$x = \cos \frac{\text{奇数} \times \pi}{2n}.$$

事实 6 $T_n(x)$ 交替地取 −1 和 1, 总共 $n+1$ 次. 事实上, 这种情形发生在 $\cos 0, \cos\frac{\pi}{n}, \cdots, \cos\frac{(n-1)\pi}{n}, \cos\pi$.

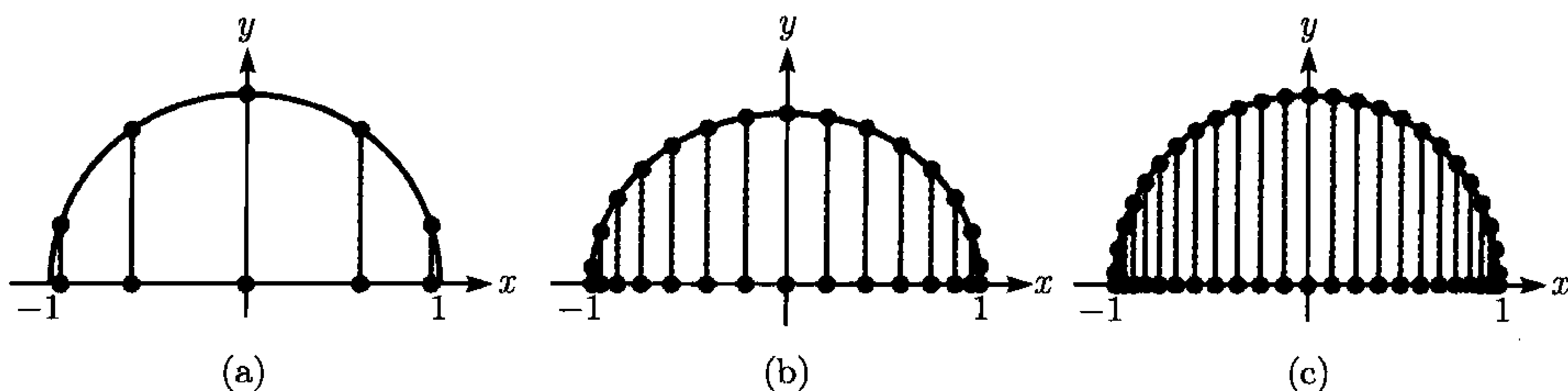


图 3-10 Chebyshev 多项式零点的位置. 这些根是圆周上等距的点的 x 坐标. (a) 5 次 (b) 15 次 (c) 25 次

从事实 2 知 $T_n(x)/2^{n-1}$ 是首项系数为 1 的多项式. 根据事实 5, 因为 $T_n(x)$ 的全部根是实数, 所以可以把 $T_n(x)/2^{n-1}$ 因式分解成 $(x-x_1)(x-x_2)\cdots(x-x_n)$, 这里 x_i 是定理 3.8 中所描述的 Chebyshev 节点.

Chebyshev 定理直接从这些事实得出.

定理 3.8 的证明. 设 $P_n(x)$ 是在 $[-1,1]$ 上具有绝对值更小的最大值且首项系数为 1 的多项式, 换言之, $|P_n(x)| < \frac{1}{2^{n-1}}, -1 \leqslant x \leqslant 1$. 这个假设导出矛盾. 因为 $T_n(x)$ 交替地取 -1 和 1, 总共 $n+1$ 次 (事实 6), 在这 $n+1$ 个点处, 差 $P_n - T_n(x)/2^{n-1}$ 交替地取正数或负数. 因此 $P_n - \frac{T_n}{2^{n-1}}$ 一定至少 n 次经过零, 也就是它必须至少有 n 个根. 因为 P_n 和 $\frac{T_n}{2^{n-1}}$ 都是首项系数为 1 的多项式, 所以它们的差的次数小于等于 $n-1$, 于是就和这个事实产生了矛盾.

3.3.3 区间的改变

到目前为止, 我们关于 Chebyshev 插值的讨论一直限制在区间 $[-1, 1]$ 中, 这是因为对这个区间, 定理 3.8 非常容易叙述. 下面我们将把整个方法推到一般的区间 $[a,b]$.

移动基点使它们在 $[a,b]$ 中与它们在 $[-1, 1]$ 中有同样的相对位置. 最好考虑分两步进行:

(1) 用因子 $(b-a)/2$(两个区间长度之比)**扩大**(stretch) 这些点的间距, 以及 (2) 用 $(b+a)/2$**平移**(translate) 这些点, 使整个中心从 0 移到 $[a,b]$ 的中点. 即, 从原来的点

$$\cos\frac{\text{奇数}\times\pi}{2n}$$

移到

$$\frac{b-a}{2}\cos\frac{\text{奇数}\times\pi}{2n}+\frac{b+a}{2}.$$

对于 $[a,b]$ 中新的 Chebyshev 基点 $x_1,\cdots,x_n$, 插值误差公式的分子中相应的上界会改变, 因为每一个因式 $(x-x_i)$ 都扩大了 $(b-a)/2$ 倍. 结果, 最小值 $1/2^{n-1}$ 必须被 $[(b-a)/2]^n/2^{n-1}$ 所代替.

Chebyshev 插值节点

在区间 $[a,b]$ 上,

$$x_i = \frac{b+a}{2} + \frac{b-a}{2}\cos\frac{(2i-1)\pi}{2n}$$

对 $i=1,\cdots,n$ 成立. 不等式

$$|(x-x_1)\cdots(x-x_n)| \leqslant \frac{\left(\frac{b-a}{2}\right)^n}{2^{n-1}} \tag{3.20}$$

在 [a, b] 上成立.

亮点 压缩

如本节所介绍的, 为了方便计算, Chebyshev 插值是把一般函数转变成少量浮点运算的好方法. 容易获得的误差上界通常比等距插值的小, 而且能够按要求设计得尽可能小.

虽然我们已经用正弦函数说明了这个过程, 但是大多数计算器及套装软件都采取不同的方法来构造实际的 "正弦键". 正弦函数的特殊性质允许用稍作改变以考虑舍入的简单的 Taylor 展开式来近似它. 因为正弦是奇函数, 所以它的 Taylor 级数在零周围的偶数项不出现, 这使得计算特别有效.

下面的例子说明 Chebyshev 插值在一般区间的应用.

例 3.11 对于区间 $[0, \pi/2]$ 上的插值, 求 4 个 Chebyshev 基点, 并且求 $f(x) = \sin x$ 在该区间上 Chebyshev 插值误差的上界.

这是第二次尝试. 在例 3.7 中我们采用等距的基点. Chebyshev 基点是

$$\frac{\frac{\pi}{2}-0}{2}\cos\left(\frac{\text{奇数}\pi}{2(4)}\right) + \frac{\frac{\pi}{2}+0}{2},$$

或者

$$x_1 = \frac{\pi}{4} + \frac{\pi}{4}\cos\frac{\pi}{8}, \quad x_2 = \frac{\pi}{4} + \frac{\pi}{4}\cos\frac{3\pi}{8}, \quad x_3 = \frac{\pi}{4} + \frac{\pi}{4}\cos\frac{5\pi}{8}, \quad x_4 = \frac{\pi}{4} + \frac{\pi}{4}\cos\frac{7\pi}{8}.$$

根据 (3.20), 对于 $0 \leqslant x \leqslant \frac{\pi}{2}$, 最坏情形的插值误差是

$$|\sin x - P_3(x)| = \frac{|(x-x_1)(x-x_2)(x-x_3)(x-x_4)|}{4!}|f''''(c)| \leqslant \frac{\left(\frac{\frac{\pi}{2}-0}{2}\right)^4}{4! \times 2^3} \times 1 \approx 0.001\,98.$$

在表 3-6 的几点处, 计算这个例子的 Chebyshev 插值多项式. 插值误差要大大低于最坏情形的估计. 图 3-11 画出了在区间 $[0, \frac{\pi}{2}]$ 上作为 x 的函数的插值误差, 并与同样的等距插值作比较. Chebyshev 误差 (虚曲线) 是小了一点, 而且在整个插值区间分布更均匀. ◀

表 3-6

x	$\sin x$	$P_3(x)$	误差
1	0.841 5	0.840 8	0.000 7
2	0.909 3	0.909 7	0.000 4
3	0.141 1	0.142 0	0.000 9
4	−0.756 8	−0.755 5	0.001 3
14	0.990 6	0.991 7	0.001 1
1 000	0.826 9	0.826 1	0.000 8

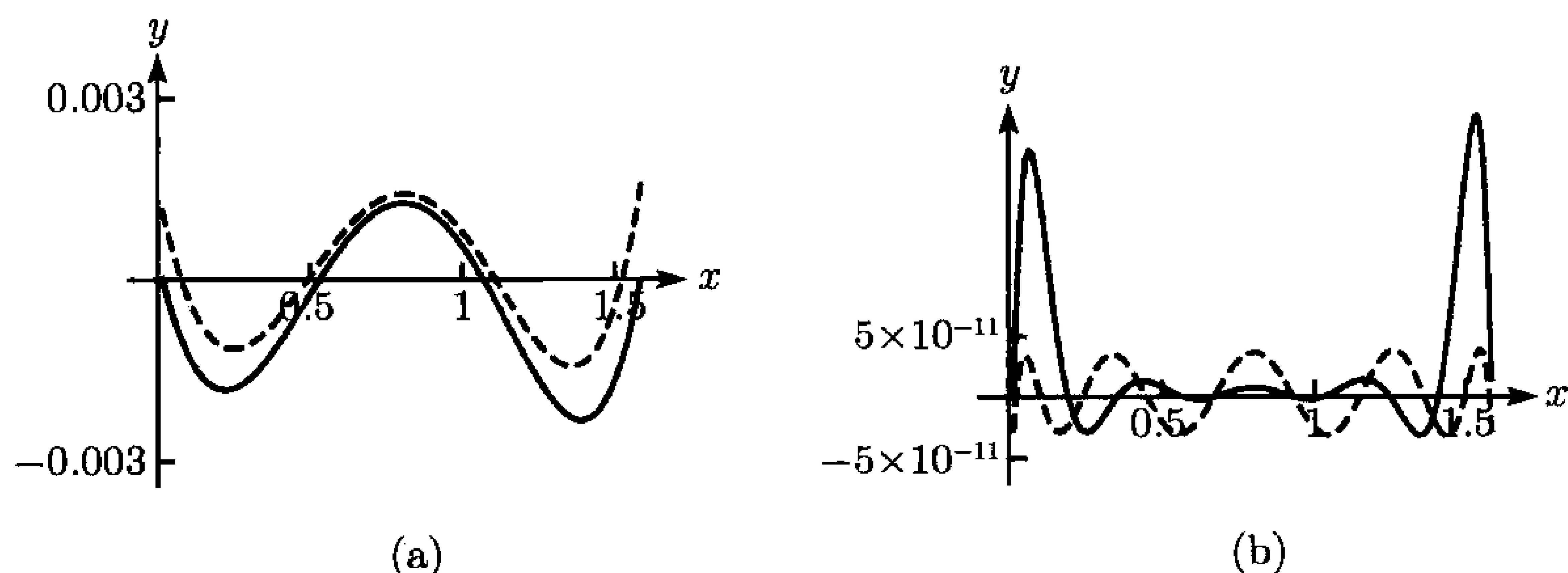


图 3-11 关于近似 $f(x) = \sin x$ 的插值误差: (a) 等距基点的三次插值多项式 (实曲线) 和 Chebyshev 基点的三次插值多项式 (虚曲线) 的插值误差, (b) 与 (a) 相同, 但是九次插值多项式

例 3.12 设计出输出会精确到 10 位小数的正弦键.

借助之前建立的正弦函数的基本定义域, 我们可以继续考虑区间 $[0, \frac{\pi}{2}]$. 重复前面的计算, 但把基点数 n 留作待定的未知数. 在区间 $[0, \frac{\pi}{2}]$ 上多项式 $P_{n-1}(x)$ 的最大插值误差是

$$|\sin x - P_{n-1}(x)| = \frac{|(x-x_1)\cdots(x-x_n)|}{n!}|f^{(n)}(c)| \leqslant \frac{\left(\frac{\frac{\pi}{2}-0}{2}\right)^n}{n! \times 2^{n-1}}.$$

这个不等式不能简单地解出 n, 但通过几次试验就会发现, 对 $n = 9$, 误差界约等于 $0.122\,4 \times 10^{-8}$, 而对 $n = 10$, 它约等于 $0.480\,7 \times 10^{-10}$, 后者满足我们关于 10 位准确小数的准则. 图 3-11b 把 Chebyshev 插值多项式的实际误差与等距插值多项式的误差进行了比较.

在 $[0, \frac{\pi}{2}]$ 中的 10 个 Chebyshev 基点是 $\pi/4 + (\pi/4)\cos(\text{奇数}\pi/20)$. 通过存储正弦函数在基点的 10 个 y 值, 并且对每一次按键做一次嵌套乘法计算就能够设计出这个键. ◀

以下的 MATLAB代码sin2.m执行了前面的任务. 这段代码写出来有一些棘手: 为了建立插值多项式在某一点的近似正弦, 我们在 10 个 Chebyshev 节点必须做 10 次正弦计算. 当然, 在实际执行中, 这些数应计算一次并且存储起来.

```
%Program 3.4 Building a sin calculator key, attempt #2
%Approximates sin curve with degree 9 polynomial
%Input:  x
%Output: approximation for sin(x), correct to 10 decimal places
function y=sin2(x)
%First calculate the interpolating polynomial and
%   store coefficients
n=10;
b=pi/4+(pi/4)*cos((1:2:2*n-1)*pi/(2*n));
yb=sin(b);                            % b holds Chebyshev base points
c=newtdd(b,yb,n);
%For each input x, move x to the fundamental domain and evaluate
%   the interpolating polynomial
s=1;                                  % Correct the sign of sin
x1=mod(x,2*pi);
if x1>pi
  x1 = 2*pi-x1;
  s = -1;
end
if x1 > pi/2
  x1 = pi-x1;
end
y = s*nest(n-1,c,x1,b);
```

本章, 为了近似三角函数, 我们经常演示等距或者 Chebyshev 节点的多项式插值. 尽管能用多项式插值近似正弦和余弦到任意的精度, 但大多数计算器还是使用一种稍微更有效的称为 CORDIC(Coordinate Rotation Digital Computer) 算法的方法 [9]. CORDIC 是基于复运算的一种精巧的迭代方法, 它能应用于几种特殊函数. 多项式插值仍然是近似一般函数以及表示和压缩数据的一种简单且有用的技术.

习题 3.3

1. 列出在给定区间中的 Chebyshev 插值节点 $x_1, \cdots, x_n$:
   (a) $[-1,1]$; $n=6$; **(b)** $[-2, 2]$; $n=4$; **(c)** $[4, 12]$; $n=6$; **(d)** $[-0.3, 0.7]$; $n=5$.
2. 对于习题 1 中的区间和 Chebyshev 节点, 求 $|(x_-x_1)\cdots(x-x_n)|$ 的上界.
3. 假设 Chebyshev 插值用于对函数 $f(x)=\mathrm{e}^x$ 在区间 $[-1, 1]$ 上求一个五次插值多项式 $Q_5(x)$. 用插值误差公式找出对于整个区间 $[-1, 1]$ 中的 x 都有效的误差 $|\mathrm{e}^x - Q_5(x)|$ 的最坏情形的估计. 当用 $Q_5(x)$ 近似 e^x 时, 小数点后面多少位数字将是准确的?
4. 对区间 $[0.6, 1.0]$, 回答习题 3 中同样的问题.
5. 当用三次 Chebyshev 插值多项式近似 $f(x)=\sin x$ 时, 求在 $[0, 2]$ 上误差的上界.

6. 假设要用 Chebyshev 插值去求区间 [3, 4] 上近似 $f(x) = x^{-3}$ 的三次插值多项式 $Q_3(x)$. **(a)** 写出用作 Q_3 的插值节点的点 (x, y). **(b)** 求对于区间 [3, 4] 中所有的 x 都有效的误差 $\left|x^{-3} - Q_3(x)\right|$ 的最坏情形的估计. 当用 $Q_3(x)$ 近似 x^{-3} 时, 小数点后面多少位数字是准确的?
7. 假定在为计算器设计显示小数点右面 6 位数字的 ln 键, 求在区间 [1, e] 上的 Chebyshev 插值将在这个精度内近似的最低次数 d.
8. 设 $T_n(x)$ 表示 n 次 Chebyshev 多项式, 求 $T_n(0)$ 的公式.
9. 确定以下各值. **(a)** $T_{999}(-1)$; **(b)** $T_{1\ 000}(-1)$; **(c)** $T_{999}(0)$; **(d)** $T_{1\ 000}(0)$; **(e)** $T_{999}(-1/2)$; **(f)** $T_{1\ 000}(-1/2)$.

计算机问题 3.3

1. 重新建立程序 3.3, 实现在区间 $[0, \frac{\pi}{2}]$ 上 4 个节点的 Chebyshev 插值多项式 (仅需改变代码中的一行), 然后画出在区间 [−2, 2] 上的多项式和正弦函数.
2. 建立 MATLAB程序, 用 Chebyshev 插值计算余弦函数, 准确到 10 位小数. 从在基本定义域 $[0, \frac{\pi}{2}]$ 插值开始, 再把你的回答推广到 $-10^4 \sim 10^4$ 的输入. 你可能要用本章中已有的 MATLAB代码.
3. 对于 $10^{-4} \sim 10^4$ 的输入, 对 $\ln x$ 执行计算机问题 2 中的步骤. 把 [1, e] 用作基本定义域. 保证 10 位准确数字的插值多项式的次数是什么? 你的程序应该从求整数 k 使得 $e^k \leqslant x \leqslant e^{k+1}$ 开始. 于是 xe^{-k} 落在基本定义域内. 通过比较你的程序与 MATLAB的`log`命令, 说明程序的精度.
4. 设 $f(x) = e^{|x|}$, 通过画出在区间 [−1, 1] 上的等距插值和 Chebyshev 插值的 n 次多项式 (n=10 及 n=20) 来比较这两种类型的插值. 对于等距插值, 左边和右边的插值基点应该是 −1 和 1. 通过用 0.01 的步长作实例试验, 每一种类型都会产生实验插值误差, 画出比较图. 在这个问题中能观察到 Runge 现象吗?
5. 对 $f(x) = e^{-x^2}$ 执行计算机问题 4 中的步骤.

3.4 三次样条

样条相当于数据插值的另一种方法. 在多项式插值中, 由多项式给出的单个公式通过所有的数据点. 样条的想法是用几个公式, 每一个是经过数据点的低次多项式.

最简单的样条例子是线性样条, 其中以直线段连结各点. 假设给定一组数据点 $(x_1, y_1), \cdots, (x_n, y_n)$, 其中 $x_1 < \cdots < x_n$. 线性样条由 $n-1$ 条直线段组成, 它们连结相邻的一对点. 图 3-12a 展示了线性样条, 在每对相邻点 $(x_i, y_i), (x_{i+1}, y_{i+1})$ 之间, 画出了通过这两点的线性函数 $y = a_i + b_i x$. 图中给出的数据点是 (1, 2), (2, 1), (4, 4), (5, 3), 而且线性样条由下式给出:

$$S_1(x) = 2 - (x - 1), \qquad x \in [1, 2],$$

$$S_2(x)=1+\frac{3}{2}(x-2),\qquad x\in[2,4],$$
$$S_3(x)=4-(x-4),\qquad x\in[4,5].\tag{3.21}$$

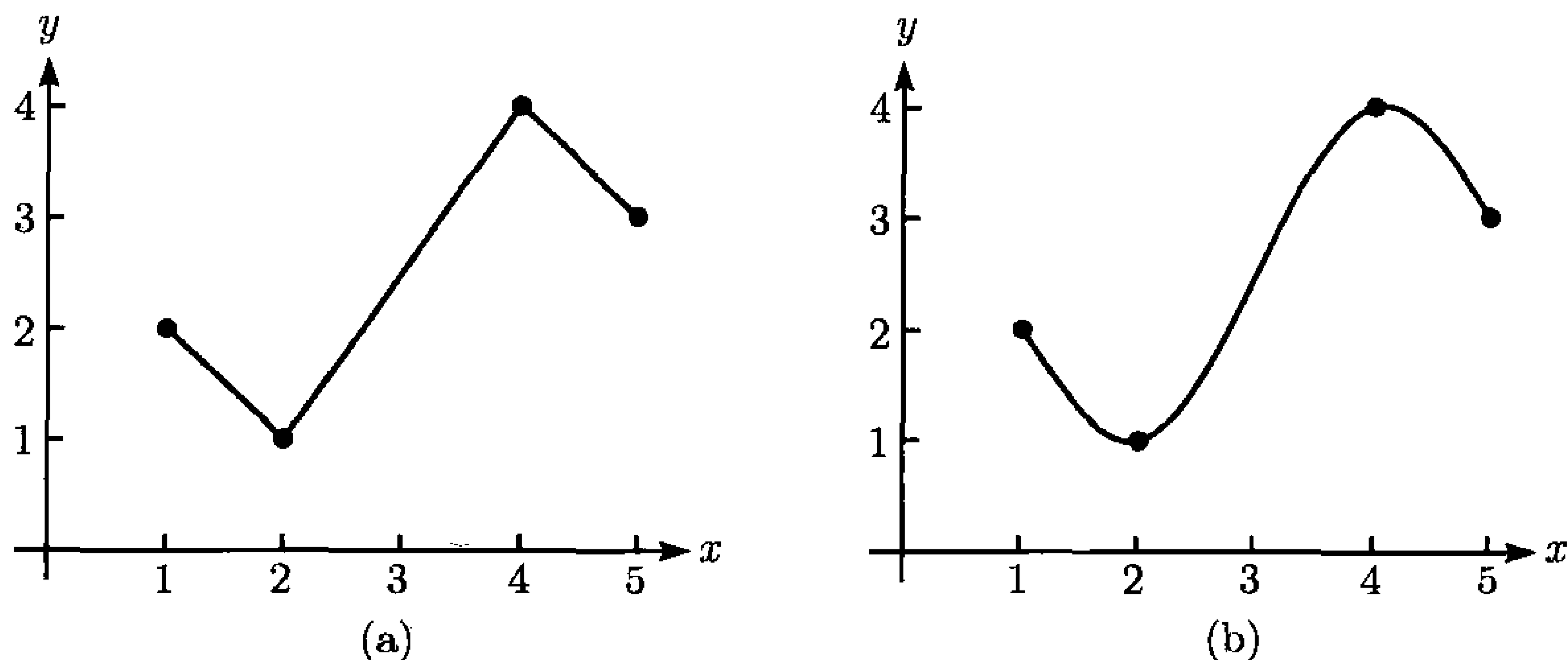


图 3-12 经过 4 个数据点的样条: (a) 经过 (1, 2), (2, 1), (4, 4), (5, 3) 的线性样条, 包括由 (3.21) 给出的 3 个线性多项式; (b) 由 (3.22) 给出的经过相同的点的三次样条

线性样条成功地插值任意一组 n 个数据点. 但是, 线性样条缺少光滑性 (smoothness). 三次样条可以克服线性样条的这种缺点. 用三次多项式, 三次样条代替了在数据点之间的线性函数.

在同样的点 (1, 2), (2, 1), (4, 4), (5, 3) 处进行插值的三次样条的例子如图 3-12b 所示. 定义样条的方程是

$$S_1(x)=2-\frac{13}{8}(x-1)+0(x-1)^2+\frac{5}{8}(x-1)^3,\quad x\in[1,2],$$
$$S_2(x)=1+\frac{1}{4}(x-2)+\frac{15}{8}(x-2)^2-\frac{5}{8}(x-2)^3,\quad x\in[2,4],$$
$$S_3(x)=4+\frac{1}{4}(x-4)-\frac{15}{8}(x-4)^2+\frac{5}{8}(x-4)^3,\quad x\in[4,5].\tag{3.22}$$

特别注意在基点或 "结点"($x=2$ 及 $x=4$) 处从一条 S_i 到下一条曲线的光滑过渡. 这是通过安排样条的相邻段 S_i 和 S_{i+1} 在结点处求值时有相同的零阶、一阶及二阶导数而达到的. 至于如何做到这一点是 3.4.1 节的内容.

给定 n 个点 $(x_1,y_1),\cdots,(x_n,y_n)$, 显然有且仅有一个线性样条经过这些数据点. 对于三次样条, 这一点将不再正确. 我们会发现有无穷多个三次样条经过任一组数据点. 在必须钉住一个特定的样条时, 将加上额外条件.

3.4.1 样条的性质

为了使三次样条的性质更确切, 我们作出以下定义: 假设给定 n 个数据点 $(x_1,y_1),\cdots,(x_n,y_n)$, 这里 x_i 互不相同而且按递增次序. 经过数据点 $(x_1,y_1),\cdots,$

(x_n, y_n) 的三次样条 $S(x)$ 是一组三次多项式

$$
\begin{aligned}
S_1(x) &= y_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3, \quad x \in [x_1, x_2], \\
S_2(x) &= y_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3, \quad x \in [x_2, x_3], \\
&\vdots \\
S_{n-1}(x) &= y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + \\
&\quad d_{n-1}(x - x_{n-1})^3, \quad x \in [x_{n-1}, x_n],
\end{aligned}
\tag{3.23}
$$

它具有以下性质:

性质 1 $S_i(x_i) = y_i, S_i(x_{i+1}) = y_{i+1},\ i = 1, \cdots, n-1.$

性质 2 $S'_{i-1}(x_i) = S'_i(x_i),\ i = 2, \cdots, n-1.$

性质 3 $S''_{i-1}(x_i) = S''_i(x_i),\ i = 2, \cdots, n-1.$

性质 1 保证样条 $S(x)$ 插值这些数据点, 性质 2 使得样条的相邻部分的斜率在它们相遇的地方一致, 而性质 3 使得由二阶导数表示的曲率相同.

例 3.13 检验 (3.22) 中的 $\{S_1, S_2, S_3\}$ 对数据点 (1, 2), (2, 1), (4, 4), (5, 3) 满足所有的样条性质.

我们将检验所有 3 个性质.

性质 1 有 n=4 个数据点, 我们必须检验

$$
\begin{aligned}
S_1(1) = 2 \quad &及 \quad S_1(2) = 1, \\
S_2(2) = 1 \quad &及 \quad S_2(4) = 4, \\
S_3(4) = 4 \quad &及 \quad S_3(5) = 3.
\end{aligned}
$$

这些容易从定义方程 (3.22) 中得出.

性质 2 样条函数的一阶导数是

$$
\begin{aligned}
S'_1(x) &= -\frac{13}{8} + \frac{15}{8}(x-1)^2, \\
S'_2(x) &= \frac{1}{4} + \frac{15}{4}(x-2) - \frac{15}{8}(x-2)^2, \\
S'_3(x) &= \frac{1}{4} - \frac{15}{4}(x-4) + \frac{15}{8}(x-4)^2.
\end{aligned}
$$

我们必须检验 $S'_1(2) = S'_2(2)$ 及 $S'_2(4) = S'_3(4)$. 第一式是

$$
\frac{-13}{8} + \frac{15}{8} = \frac{1}{4},
$$

而第二式是

$$
\frac{1}{4} + \frac{15}{4}(4-2) - \frac{15}{8}(4-2)^2 = \frac{1}{4},
$$

两个都检验完毕.

性质 3 二阶导数是

$$
\begin{aligned}
S_1''(x) &= \frac{15}{4}(x-1), \\
S_2''(x) &= \frac{15}{4} - \frac{15}{4}(x-2), \\
S_3''(x) &= \frac{-15}{4} + \frac{15}{4}(x-4).
\end{aligned} \tag{3.24}
$$

我们必须检验 $S_1''(2) = S_2''(2)$ 及 $S_2''(4) = S_3''(4)$, 两个都是正确的, 因此 (3.22) 是三次样条. ◀

从一组数据点构造样条就是求使得性质 1~3 成立的系数 b_i, c_i, d_i. 在讨论如何确定样条的未知系数 b_i, c_i, d_i 之前, 让我们清点一下由定义施加的条件数. 性质 1 的前半部分已经在 (3.23) 中得到反映; 它说明三次多项式 S_i 的常数项必须是 y_i. 性质 1 的后半部分包括 $n-1$ 个独立的方程, 其被我们考虑为未知量的系数必须得到满足. 性质 2 和性质 3 各自加上 $n-2$ 个方程, 总共是 $n-1+2(n-2) = 3n-5$ 个要满足的独立方程.

有多少个未知系数? 对于样条的每一部分 S_i, 需要 3 个系数 b_i, c_i, d_i, 总共是 $3(n-1) = 3n-3$. 因此, 求解系数是一个求解含 $3n-3$ 个未知量的 $3n-5$ 个线性方程的问题. 除非在方程组中存在不相容的方程 (实际上并没有), 否则这个方程组就是亚定方程组, 因此有无穷多解. 换言之, 有无穷多条三次样条经过这组任意的数据点 $(x_1, y_1), \cdots, (x_n, y_n)$.

样条使用者一般通过添加两个额外的方程到这 $3n-5$ 个方程中, 以得到含 m 个未知量的 m 个方程的方程组, 这里 $m = 3n-3$, 从而克服这个缺点. 除了允许使用者限制样条于给定的详细说明外, 把范围缩小到一个解简化了计算和描述结果.

添加另外两个约束的简单方法不仅需要前面的 $3n-5$ 个约束, 还需要样条 $S(x)$ 在定义区间的每个端点有一个反射点. 加到性质 1~3 上的约束是

性质 4a (自然样条) $S_1''(x_1) = 0$ 及 $S_{n-1}''(x_n) = 0$.

满足这两个附加条件的三次样条称为**自然三次样条**. 注意, (3.22) 是自然三次样条, 这是因为从 (3.24) 容易验证 $S_1''(1) = 0$ 及 $S_3''(5) = 0$.

还有其他一些方法可以添加两个额外的条件. 通常, 如在自然样条的情形那样, 这些方法都要确定样条的左右端点的额外的性质, 所以称它们为**端点条件**(end condition). 我们将在 3.4.2 节讲述这个主题, 但是现在将集中在自然三次样条.

既然有了适当的方程个数, 含 $3n-3$ 个未知量的 $3n-3$ 个方程, 我们就能够写一个 MATLAB函数来求解样条系数. 首先, 写出关于未知量 b_i, c_i, d_i 的方程, 性质 1

的第二部分就意味着 $n-1$ 个方程:

$$\begin{aligned} y_2 &= S_1(x_2) = y_1 + b_1(x_2-x_1) + c_1(x_2-x_1)^2 + d_1(x_2-x_1)^3, \\ &\vdots \\ y_n &= S_{n-1}(x_n) = y_{n-1} + b_{n-1}(x_n - x_{n-1}) + \\ &\quad c_{n-1}(x_n - x_{n-1})^2 + d_{n-1}(x_n - x_{n-1})^3. \end{aligned} \tag{3.25}$$

性质 2 产生 $n-2$ 个方程:

$$\begin{aligned} 0 &= S_1'(x_2) - S_2'(x_2) = b_1 + 2c_1(x_2-x_1) + 3d_1(x_2-x_1)^2 - b_2 \\ &\vdots \\ 0 &= S_{n-2}'(x_{n-1}) - S_{n-1}'(x_{n-1}) = b_{n-2} + 2c_{n-2}(x_{n-1} - x_{n-2}) \\ &\quad +3d_{n-2}(x_{n-1} - x_{n-2})^2 - b_{n-1}, \end{aligned} \tag{3.26}$$

而性质 3 意味着 $n-2$ 个方程

$$\begin{aligned} 0 &= S_1''(x_2) - S_2''(x_2) = 2c_1 + 6d_1(x_2-x_1) - 2c_2 \\ &\vdots \\ 0 &= S_{n-2}''(x_{n-1}) - S_{n-1}''(x_{n-1}) = 2c_{n-2} + 6d_{n-2}(x_{n-1} - x_{n-2}) - 2c_{n-1}. \end{aligned} \tag{3.27}$$

代替解这种形式的方程, 通过解耦方程能够大大简化这个方程组. 用少量代数知识, 可以先解有关 c_i 的简单得多的方程组, 接着求出用已知的 c_i 表示的对于 b_i 和 d_i 的显式公式.

如果引入额外的未知量 $c_n = S_{n-1}''(x_n)/2$, 这在概念上就会比较简单. 此外, 我们引入简化符号 $\delta_i = x_{i+1} - x_i$ 及 $\Delta_i = y_{i+1} - y_i$, 那么 (3.27) 就能解出系数

$$d_i = \frac{c_{i+1} - c_i}{3\delta_i}, \quad i = 1, \cdots, n-1. \tag{3.28}$$

从 (3.25) 解出 b_i 得到

$$\begin{aligned} b_i &= \frac{\Delta_i}{\delta_i} - c_i\delta_i - d_i\delta_i^2 = \frac{\Delta_i}{\delta_i} - c_i\delta_i - \frac{\delta_i}{3}(c_{i+1} - c_i) \\ &= \frac{\Delta_i}{\delta_i} - \frac{\delta_i}{3}(2c_i + c_{i+1}), \quad i = 1, \cdots, n-1. \end{aligned} \tag{3.29}$$

把 (3.28) 和 (3.29) 代入 (3.26) 得到以下 $n-2$ 个关于 $c_1, \cdots, c_n$ 的方程:

$$\delta_1 c_1 + 2(\delta_1+\delta_2)c_2 + \delta_2 c_3 = 3\left(\frac{\Delta_2}{\delta_2} - \frac{\Delta_1}{\delta_1}\right),$$

$$\vdots$$

$$\delta_{n-2} c_{n-2} + 2(\delta_{n-2}+\delta_{n-1})c_{n-1} + \delta_{n-1} c_n = 3\left(\frac{\Delta_{n-1}}{\delta_{n-1}} - \frac{\Delta_{n-2}}{\delta_{n-2}}\right).$$

两个增加的方程由自然样条条件 (性质 4a) 给出:

$$S_1''(x_1) = 0 \to 2c_1 = 0,$$
$$S_{n-1}''(x_n) = 0 \to 2c_n = 0.$$

这就给出 n 个未知量 c_i 的总共 n 个方程, 可以写成矩阵形式:

$$\begin{bmatrix} 1 & 0 & 0 & & & \\ \delta_1 & 2\delta_1+2\delta_2 & \delta_2 & \ddots & & \\ 0 & \delta_2 & 2\delta_2+2\delta_3 & \delta_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & & \delta_{n-2} & 2\delta_{n-2}+2\delta_{n-1} & \delta_{n-1} \\ & & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ \\ \vdots \\ \\ c_n \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 3\left(\frac{\Delta_2}{\delta_2} - \frac{\Delta_1}{\delta_1}\right) \\ \vdots \\ 3\left(\frac{\Delta_{n-1}}{\delta_{n-1}} - \frac{\Delta_{n-2}}{\delta_{n-2}}\right) \\ 0 \end{bmatrix} \tag{3.30}$$

由 (3.30) 确定 $c_1, \cdots, c_n$, 再从 (3.28) 及 (3.29) 求出 $b_1, \cdots, b_{n-1}$ 及 $d_1, \cdots, d_{n-1}$.

注意 (3.30) 对于 c_i 永远可解. 它的系数矩阵严格对角占优, 所以根据第 2 章的定理 2.9, 对于 c_i 存在唯一解, 对 b_i 和 c_i 也是这样. 于是证明了下面的定理.

定理 3.9 设 $n \geqslant 2$, 对于一组数据点 $(x_1, y_1), \cdots, (x_n, y_n)$, 其中 x_i 互不相同, 存在唯一的自然三次样条拟合这些点.

自然三次样条

给定 $x = [x_1, \cdots, x_n]$, 这里 $x_1 < \cdots < X_n, y = [y_1, \cdots, y_n]$

for $i = 1, \cdots, n-1$

$a_i = y_i$

$\delta_i = x_{i+1} - x_i$

$$\Delta_i = y_{i+1} - y_i$$

end

求解 (3.30) 得到 $c_1, \cdots, c_n$

for $i = 1, \cdots, n-1$

$$d_i = \frac{c_{i+1} - c_i}{3\delta_i}$$

$$b_i = \frac{\Delta_i}{\delta_i} - \frac{\delta_i}{3}(2c_i + c_{i+1})$$

end

在区间 $[x_i, x_{i+1}]$ 上的自然三次样条是

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad i = 1, \cdots, n-1.$$

例 3.14 求经过 (0, 3), (1, −2), (2, 1) 的自然三次样条.

x 坐标是 $x_1 = 0, x_2 = 1, x_3 = 2$. y 坐标是 $a_1 = y_1 = 3, a_2 = y_2 = -2, a_3 = y_3 = 1$, 并且差是 $\delta_l = \delta_2 = 1, \Delta_1 = -5$, $\Delta_2 = 3$. 三对角矩阵方程 (3.30) 是

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 4 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 24 \\ 0 \end{bmatrix}.$$

它的解是 $[c_1, c_2, c_3] = [0, 6, 0]$. 现在, 由 (3.28) 及 (3.29) 得到

$$\begin{aligned} d_1 &= \frac{c_2 - c_1}{3\delta_1} = \frac{6}{3} = 2, \\ d_2 &= \frac{c_3 - c_2}{3\delta_2} = \frac{-6}{3} = -2, \\ b_1 &= \frac{\Delta_1}{\delta_1} - \frac{\delta_1}{3}(2c_1 + c_2) = -5 - \frac{1}{3}(6) = -7, \\ b_2 &= \frac{\Delta_2}{\delta_2} - \frac{\delta_2}{3}(2c_2 + c_3) = 3 - \frac{1}{3}(12) = -1. \end{aligned}$$

因此, 三次样条是

$$\begin{aligned} S_1(x) &= 3 - 7x + 0x^2 + 2x^3, x \in [0, 1]. \\ S_2(x) &= -2 - 1(x - 1) + 6(x - 1)^2 - 2(x - 1)^3, x \in [1, 2]. \end{aligned}$$

这种计算的 MATLAB代码如下: 对于不同的 (不是自然的) 端点条件 (在 3.4.2 节中讨论), (3.30) 的第一行和最后一行被其他合适的行代替.

```
%Program 3.5 Calculation of spline coefficients
%Calculates coefficients of cubic spline
%Input: x,y vectors of data points
%   plus two optional extra data v1, vn
%Output: matrix of coefficients b1,c1,d1;b2,c2,d2;...
function coeff=splinecoeff(x,y)
n=length(x);v1=0;vn=0;
A=zeros(n,n);           % matrix A is nxn
r=zeros(n,1);
for i=1:n-1             % define the deltas

    dx(i)= x(i+1)-x(i); dy(i)=y(i+1)-y(i);
end
for i=2:n-1             % load the A matrix
    A(i,i-1:i+1)=[dx(i-1) 2*(dx(i-1)+dx(i)) dx(i)];
    r(i)=3*(dy(i)/dx(i)-dy(i-1)/dx(i-1)); % right-hand side
end
% Set endpoint conditions
% Use only one of following 5 pairs:
A(1,1) = 1;             % natural spline conditions
A(n,n) = 1;
%A(1,1)=2;r(1)=v1;      % curvature-adj conditions
%A(n,n)=2;r(n)=vn;
%A(1,1:2)=[2*dx(1) dx(1)];r(1)=3*(dy(1)/dx(1)-v1);  %clamped
%A(n,n-1:n)=[dx(n-1) 2*dx(n-1)];r(n)=3*(vn-dy(n-1)/dx(n-1));
%A(1,1:2)=[1 -1];       % parabol-term conditions, for n>=3
%A(n,n-1:n)=[1 -1];
%A(1,1:3)=[dx(2) -(dx(1)+dx(2)) dx(1)]; % not-a-knot, for n>=4
%A(n,n-2:n)=[dx(n-1) -(dx(n-2)+dx(n-1)) dx(n-2)];
coeff=zeros(n,3);
coeff(:,2)=A\r;         % solve for c coefficients
for i=1:n-1             % solve for b and d
    coeff(i,3)=(coeff(i+1,2)-coeff(i,2))/(3*dx(i));
    coeff(i,1)=dy(i)/dx(i)-dx(i)*(2*coeff(i,2)+coeff(i+1,2))/3;
end
coeff=coeff(1:n-1,1:3);
```

我们已经取得了列出关于端点条件的其他选择的自由, 尽管现在没有说明它们. 在 3.4.2 节将讨论其他可能的条件. 另一个 MATLAB函数命名为`splineplot.m`, 调用`splinecoeff.m`可以得到系数并画出三次样条.

```
%Program 3.6 Cubic spline plot
%Plots spline from data points
%Input: x,y vectors of data points
%Output: none
function splineplot(x,y)
n=length(x); coeff=splinecoeff(x,y);
clf;hold on;
for i=1:n-1             % clear figure window and turn hold on
```

```
    x0=linspace(x(i),x(i+1),100);
    dx=x0-x(i);
    y0=coeff(i,3)*dx; % evaluate using nested multiplication
    y0=(y0+coeff(i,2)).*dx;
    y0=(y0+coeff(i,1)).*dx+y(i);
    plot([x(i) x(i+1)],[y(i) y(i+1)],'o',x0,y0)
end
hold off
```

图 3-13a 描述了用`splinecoeff.m.`生成的自然三次样条.

3.4.2 端点条件

性质 4a 中确定的两个额外条件称为自然样条的"端点条件". 要求这些条件与性质 1~3 一起被满足, (按照定理 3.9) 把范围缩小到恰好一个三次样条. 这说明有性质 4 的许多不同形式, 即许多其他端点条件, 对此, 一个类似的定理成立. 本节提出几个比较常见的.

性质 4b (曲率–调准三次样条) 对自然三次样条的第一个替换需要置 $S_1''(x_1)$ 和 $S_{n-1}''(x_n)$ 为使用者选取的任意非零值. 这种选取对应于在样条的左右端点设置所要的曲率. 根据 (3.23), 这就转化成两个额外条件

$$2c_1 = v_1, \qquad 2c_n = v_n,$$

这里 v_1, v_n 表示所要的值. 方程变成两行表格

$$\left[\begin{array}{cccccccc|c} 2 & 0 & 0 & 0 & 0 & \cdots\cdots & 0 & 0 & v_1 \\ 0 & 0 & 0 & 0 & 0 & \cdots\cdots & 0 & 2 & v_n \end{array}\right]$$

来代替 (3.30) 中对自然样条所加上去的第一行和最后一行. 注意新的系数矩阵也是严格对角占优, 所以对于曲率–调准样条而言定理 3.9 的一般形式成立. (见下面即将提出的定理 3.10.) 在`splinecoeff.m`中, 这两行

```
A(1,1)=2;r(1)=v1;        % curvature-adj conditions
A(n,n)=2;r(n)=vn;
```

必须替换自然样条中现有的两行.

下一个端点条件的取舍是

性质 4c (夹子三次样条) 这种选择与前一种类似, 但是, 它是一阶导数 $S_1'(x_1)$ 及 $S_{n-1}'(x_n)$, 它们被设置为使用者分别指定的值 v_1 及 v_n. 于是在样条的起点和终点的斜率就在使用者的控制之下.

利用 (3.28) 及 (3.29), 我们可以把额外条件 $S_1'(x_l) = v_1$ 写成

$$2\delta_1 c_1 + \delta_1 c_2 = 3\left(\frac{\Delta_1}{\delta_1} - v_1\right)$$

并把 $S'_{n-1}(x_n) = v_n$ 写成

$$\delta_{n-1}c_{n-1} + 2\delta_{n-1}c_n = 3\left(v_n - \frac{\Delta_{n-1}}{\delta_{n-1}}\right).$$

相应的两行表格是

$$\left[\begin{array}{cccccc|c} 2\delta_1 & \delta_1 & 0 & 0\cdots\cdots0 & 0 & 0 & 3(\Delta_1/\delta_1 - v_1) \\ 0 & 0 & 0 & 0\cdots\cdots0 & \delta_{n-1} & 2\delta_{n-1} & 3(v_n - \Delta_{n-1}/\delta_{n-1}) \end{array}\right].$$

注意, (3.30) 中修改过的系数矩阵也是严格对角占优, 因此定理 3.9 在自然样条被夹子样条代替后也成立. 在splinecoeff.m中, 这两行

```
A(1,1:2)=[2*dx(1) dx(1)];r(1)=3*(dy(1)/dx(1)-v1);
A(n,n-1:n)=[dx(n-1) 2*dx(n-1)];r(n)=3*(vn-dy(n-1)/dx(n-1));
```

必须被替换. 关于取 $v_1 = v_n = 0$ 的夹子样条, 见图 3-13.

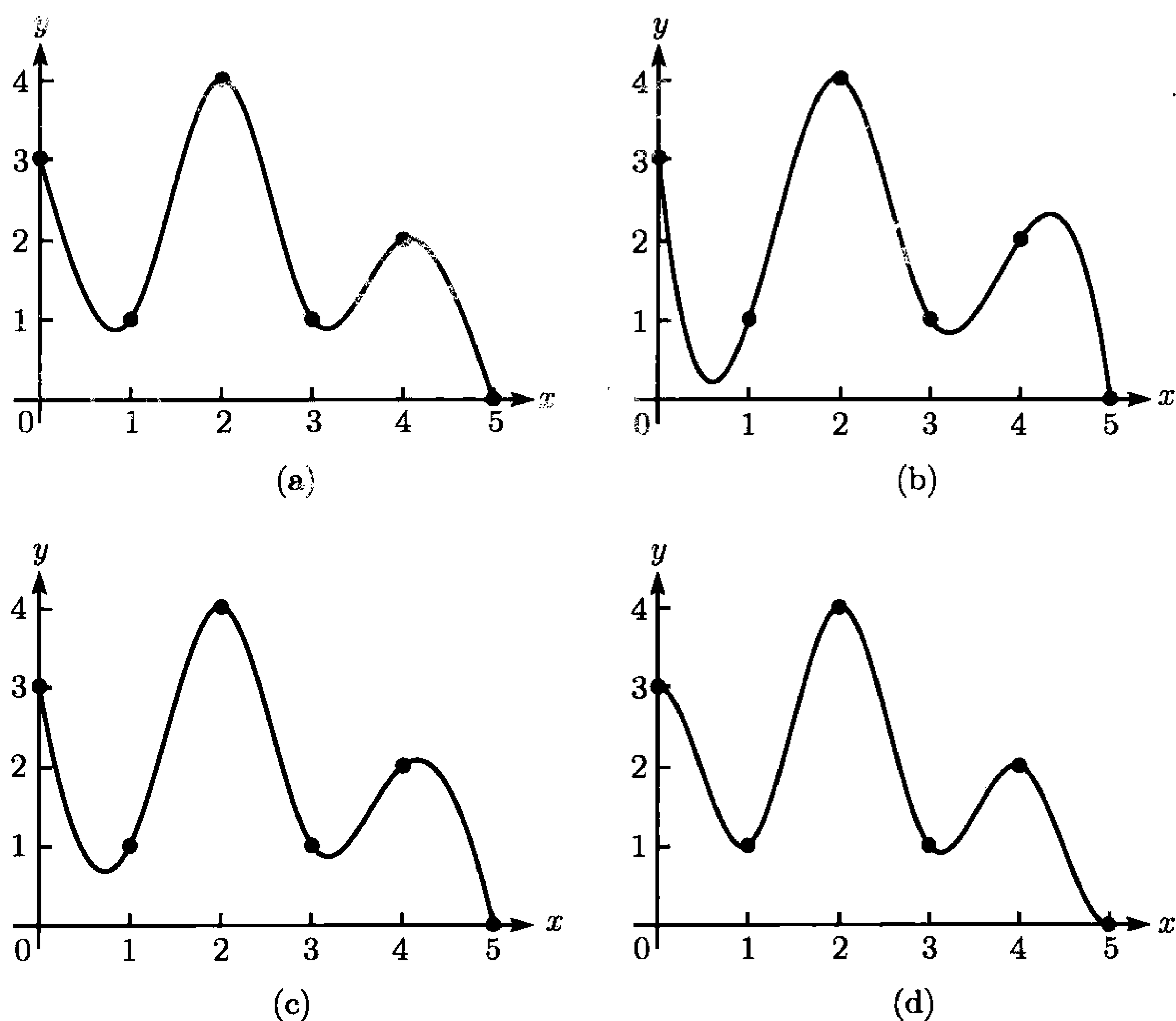


图 3-13　经过 6 个点的三次样条. 图是用splinecoeff.m以输入向量x = [0 1 2 3 4 5]及y = [3 1 4 1 2 0]生成的: (a) 自然三次样条 (注意在端点的反射点); (b) 非节点的三次样条 (在 [0, 2] 和 [3, 5] 上的三次方程); (c) 末端抛物线样条; (d) 夹子三次样条 (在两端点斜率 0 处夹住)

性质 4d (末端抛物线三次样条) 样条的第一部分和最后部分, S_1 及 S_{n-1}, 通过指定 $d_1 = 0 = d_{n-1}$ 被限制成最多是二次的. 等价地, 根据 (3.28) 我们可以要求 $c_1 = c_2$ 及 $c_{n-1} = c_n$. 这两个方程形成的两行表格

$$\left[\begin{array}{cccccccccc|c} 1 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & -1 & & 0 \end{array}\right]$$

将被用作 (3.30) 的第一行和最后一行. 假设数据点的个数 n 满足 $n \geqslant 3$. ($n = 2$ 的情形见习题 17.) 在这种情形下, 用 c_2 代替 c_1, 用 c_{n-1} 代替 c_n, 我们发现矩阵方程退化为关于 $c_2, \cdots, c_{n-1}$ 的严格对角占优 $(n-2) \times (n-2)$ 矩阵方程. 因此, 定理 3.9 的形式对于末端抛物线样条成立 (假设 $n \geqslant 3$).

在`splinecoeff.m`中, 这两行

```
A(1,1:2)=[1 -1];                % parabol-term conditions
A(n,n-1:n)=[1 -1];
```

必须被替换.

性质 4e (非节点的三次样条) 两个加进去的方程是 $d_1 = d_2$ 及 $d_{n-2} = d_{n-1}$, 或者等价地, $S_1'''(x_2) = S_2'''(x_2), S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1})$. 因为 S_1 及 S_2 是次数小于等于 3 的多项式, 要求它们的三阶导数在 x_1 处相等, 而它们的零阶、一阶及二阶导数已经在 x_1 处相等, 使得 S_1 和 S_2 是恒等的三次多项式 (三次多项式由 4 个系数确定, 因此由 4 个条件所确定), 所以并不需要 x_2 作为基点: 通过在整个区间 $[x_1, x_3]$ 上相同的公式 $S_1 = S_2$ 给出样条. 同样的推理说明 $S_{n-2} = S_{n-1}$, 所以不仅 x_2 而且 x_{n-1} 也是 "不再为节点".

注意, $d_1 = d_2$ 意味着 $(c_2 - c_1)/\delta_1 = (c_3 - c_2)/\delta_2$, 或者

$$\delta_2 c_1 - (\delta_1 + \delta_2)c_2 + \delta_1 c_3 = 0,$$

而且同样地, $d_{n-2} = d_{n-1}$ 意味着

$$\delta_{n-1}c_{n-2} - (\delta_{n-2} + \delta_{n-1})c_{n-1} + \delta_{n-2}c_n = 0.$$

这样得到的两行表格是

$$\left[\begin{array}{cccccccc|c} \delta_2 & -(\delta_1+\delta_2) & \delta_1 & 0\cdots\cdots 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0\cdots\cdots 0 & \delta_{n-1} & -(\delta_{n-2}+\delta_{n-1}) & \delta_{n-2} & & 0 \end{array}\right]$$

在`splinecoeff.m`中, 这两行

```
A(1,1:3)=[dx(2) -(dx(1)+dx(2)) dx(1)]; % not-a-knot conditions
A(n,n-2:n)=[dx(n-1) -(dx(n-2)+dx(n-1)) dx(n-2)];
```

要用到. 图 3-13b 描述了非节点三次样条的例子, 并与经过相同数据点的图 3-13a 的自然样条进行了比较.

如前面提及, 对前面每种端点条件的选取, 存在类似于定理 3.9 的定理.

定理 3.10 假设 $n \geqslant 2$. 对一组数据点 $(x_1, y_1), \cdots, (x_n, y_n)$ 以及任何一种由性质 4a ~ 4c 给出的端点条件, 存在唯一的三次样条满足端点条件并且拟合了这些点. 对于 $n \geqslant 3$ 的性质 4d, 以及对于 $n \geqslant 4$ 的性质 4e, 上述亦真.

MATLAB的默认spline命令在给出 4 个或更多个点时, 构造了非节点样条. 设x和Y分别是包含 x_i 及 y_i 数据值的向量, 那么非节点样条在另一个输入x0 处的 y 坐标通过 MATLAB命令

```
>> y0 = spline(x,y,x0);
```

算得.

如果x0是 x 坐标的向量, 那么输出y0 相应地将是 y 坐标的向量, 适用于绘图, 等等. 另外, 如果向量输入y恰好比x多两个输入, 就取夹子 v_1 和 v_n 等于y的第一个和最后一个分量, 计算这种夹子三次样条.

习题 3.4

1. 确定以下方程是否构成三次样条:

(a) $S(x) = \begin{cases} x^3 + x - 1, & x \in [0,1] \\ -(x-1)^3 + 3(x-1)^2 + 3(x-1) + 1, & x \in [1,2] \end{cases}$

(b) $S(x) = \begin{cases} 2x^3 + x^2 + 4x + 5, & x \in [0,1] \\ (x-1)^3 + 7(x-1)^2 + 12(x-1) + 12, & x \in [1,2] \end{cases}$

2. (a) 对下式检验样条条件:

$$\begin{cases} S_1(x) = 1 + 2x + 3x^2 + 4x^2, & x \in [0,1] \\ S_2(x) = 10 + 20(x-1) + 15(x-1)^2 + 4(x-1)^2, & x \in [1,2] \end{cases}$$

(b) 不管你对 (a) 的答案, 确定以下的任何额外条件是否满足这个例子: 自然、末端抛物线和非节点.

3. 在下面的三次样条中求出 c. 自然、末端抛物线和非节点, 这三种端点条件中的哪一种是满足的 (若有的话)?

(a) $S(x) = \begin{cases} 4 - \frac{11}{4}x + \frac{3}{4}x^3, & x \in [0,1] \\ 2 - \frac{1}{2}(x-1) + c(x-1)^2 - \frac{3}{4}(x-1)^3, & x \in [1,2] \end{cases}$

(b) $S(x) = \begin{cases} 3 - 9x + 4x^2, & x \in [0,1] \\ -2 - (x-1) + c(x-1)^2, & x \in [1,2] \end{cases}$

(c) $S(x) = \begin{cases} -2 - \frac{3}{2}x + \frac{7}{2}x^2 - x^3, & x \in [0,1] \\ -1 + c(x-1) + \frac{1}{2}(x-1)^2 - (x-1)^3, & x \in [1,2] \\ 1 + \frac{1}{2}(x-2) - \frac{5}{2}(x-2)^2 - (x-2)^3, & x \in [2,3] \end{cases}$

4. 在下面的三次样条中求出 k_1, k_2, k_3. 三种端点条件 (自然、末端抛物线和非节点) 中的哪一种满足的 (若有的话)?

$$S(x)=\begin{cases}4+k_1x+2x^2-\dfrac{1}{6}x^3,\ x\in[0,1]\\ 1-\dfrac{4}{3}(x-1)+k_2(x-1)^2-\dfrac{1}{6}(x-1)^3,\ x\in[1,2]\\ 1+k_3(x-2)+(x-2)^2-\dfrac{1}{6}(x-2)^3,\ x\in[2,3]\end{cases}$$

5. 对给定的数据 (0, 0), (1, 1), (2, 2), 存在多少个 [0, 2] 上的自然三次样条? 描述一个这样的样条.

6. 求经过数据点 (0, 1), (1, 1), (2, 1), (3, 1), (4, 1) 的末端抛物线三次样条. 这个样条也是非节点的吗? 是自然的吗?

7. 解方程 (3.30), 求经过以下 3 点 (a) (0, 0), (1, 1), (2, 4); (b) $(-1, 1)$, (1, 1), (2, 4) 的自然三次样条.

8. 解方程 (3.30), 求经过以下 3 点 (a) (0, 1), (2, 3), (3, 2); (b) (0, 0), (1, 1), (2, 6) 的自然三次样条.

9. 求三次样条

$$\begin{cases}S_1(x)=3+b_1x+x^3,\ x\in[0,1]\\ S_2(x)=1+b_2(x-1)+3(x-1)^2-2(x-1)^3,\ x\in[1,3]\end{cases}$$

的 $S'(0)$ 及 $S'(3)$.

10. 判断对错: 给定 $n=3$ 个数据点, 经过这些点的末端抛物线三次样条必是非节点样条.

11. **(a)** 对给定的数据 (0, 2), (1, 0), (2, 2), 有多少个 [0, 2] 上的末端抛物线三次样条?
(b) 对非节点, 回答同样的问题.

12. 对给定的数据 (1, 3), (3, 3), (4, 2), (5, 0), 有多少个非节点三次样条? 描述一个这样的样条.

13. 三次样条可以既是自然的又是末端抛物线的吗? 如果可以, 关于这样的样条你还能说些什么?

14. (同时) 存在通过一组 x_i 互不相同的数据点 $(x_1, y_1), \cdots, (x_{100}, y_{100})$ 的自然、末端抛物线和非节点的三次样条吗? 如果存在, 给出理由. 如果不存在, 解释为了使这样的样条存在, 在这 100 个点上必须满足什么条件?

15. 假设给定的自然三次样条的最左边都是常数函数 $S_1(x)=1, x\in[-1,0]$. 求在 [0, 1] 上的相邻部分 $S_2(x)$ 的 3 种不同的可能.

16. 假设一辆汽车沿着直线从一点到另一点行驶, 在时刻 $t=0$ 从一个地点出发, 在时刻 $t=1$ 到另一个地点停止. 沿着道路的距离是在 0 与 1 之间某些时刻抽样. 哪一种三次样条 (根据端点条件) 最适合描述距离与时间的关系曲线?

17. 对于末端抛物线三次样条, 情形 $n=2$ 没有被定理 3.10 覆盖. 讨论这种三次样条的存在性与唯一性.

18. 当 $n=2$ 及 $n=3$ 时, 讨论非节点三次样条的存在性和唯一性.

计算机问题 3.4

1. 求插值数据点**(a)** (0, 3), (1, 5), (2, 4), (3, 1), **(b)** (−1, 3), (0, 5), (3, 1), (4, 1), (5, 1) 的自然三次样条的方程并画图.
2. 求插值数据点**(a)** (0, 3), (1, 5), (2, 4), (3, 1), **(b)** (−1, 3), (0, 5), (3, 1), (4, 1), (5, 1) 的非节点三次样条并画图.
3. 求满足 $S(0) = 1, S(1) = 3, S(2) = 3, S(3) = 4, S(4) = 2$ 以及 $S''(0) = S''(4) = 0$ 的三次样条 S 并画图.
4. 求满足 $S(0) = 1, S(1) = 3, S(2) = 3, S(3) = 4, S(4) = 2$ 以及 $S''(0) = 3$, $S''(4) = 2$ 的三次样条 S 并画图.
5. 求满足 $S(0) = 1, S(1) = 3, S(2) = 3, S(3) = 4, S(4) = 2$ 以及 $S'(0) = 0$, $S'(4) = 1$ 的三次样条 S 并画图.
6. 求满足 $S(0) = 1, S(1) = 3, S(2) = 3, S(3) = 4, S(4) = 2$ 以及 $S'(0) = -2$, $S'(4) = 1$ 的三次样条 S 并画图.
7. 求在 $[0, \frac{\pi}{2}]$(包括端点) 中的 5 个等距的点插值 $f(x) = \cos x$ 的夹子三次样条. 为了极小化插值误差, $S'(0)$ 及 $S'(\pi \div 2)$ 的最佳选择是什么? 画出在 [0, 2] 上的样条及 $\cos x$.
8. 对函数 $f(x) = \sin x$, 执行计算机问题 7 中的步骤.
9. 求在 [1, 3](包括端点) 中的 5 个等距的点插值 $f(x) = \ln x$ 的夹子三次样条. 以实验为基础求在 [1, 3] 上最大的插值误差.
10. 在计算机问题 9 中要求产生的最大插值误差最多是 0.5×10^{-7}, 求插值结点的个数.
11. **(a)** 考虑经过在计算机问题 3.1.1 中的世界人口数据点的自然三次样条. 计算 1980 年的值, 并且与准确的人口数比较. **(b)** 用线性样条估计在 1960 年及 2000 年的斜率, 并用这些斜率求经过这些数据点的夹子三次样条. 画出样条并估计 1980 年的人口. 哪一种估计更好, 自然的还是夹子的?
12. 回忆习题 3.1.13 的二氧化碳数据. **(a)** 求经过这些数据点的自然三次样条并画图, 为 1950 年的二氧化碳浓度计算这个样条估计. **(b)** 对末端抛物样条执行同样的分析. **(c)** 非节点样条如何不同于习题 3.1.13 的解?
13. 在单个图中, 描述经过计算机问题 3.2.3 中世界石油产量数据点的自然、非节点以及末端抛物线三次样条.

3.5 Bézier 曲线

Bézier 曲线是允许用户控制在节点处斜率的样条. 作为这种额外自由的代价, 是不再保证 3.4 节中的三次样条自动具有的特性, 即经过节点的一阶及二阶导数的光滑性. Bézier 样条适用于以下偶而会需要的场合: 有角 (一阶导数不连续) 以及曲率急剧变化 (二阶导数不连续).

Pierre Bézier 为雷诺汽车公司工作时产生了这种想法. 在另一家竞争的汽车公司雪铁龙工作的 Paul de Casteljau 也独立地发现了同样的想法. 它被两家公司当

作商业秘密, 而且当 Bézier 发表他的研究后, 大家才知道原来两人英雄所见略同. 今天, Bézier 曲线已成为计算机辅助设计和制造的基础.

每一段平面 Bézier 样条由 4 个点 (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) 确定. 第一个点和最后一个点是样条曲线的端点, 而中间两个点是**控制点**(control point), 如图 3-14 所示. 曲线沿着切线方向 $(x_2 - x_1, y_2 - y_1)$ 从 (x_1, y_1) 出发, 并且沿着切线方向 $(x_4 - x_3, y_4 - y_3)$ 在 (x_4, y_4) 终止. 完成这一切的方程表示为参数曲线 $(x(t), y(t)), 0 \leqslant t \leqslant 1$.

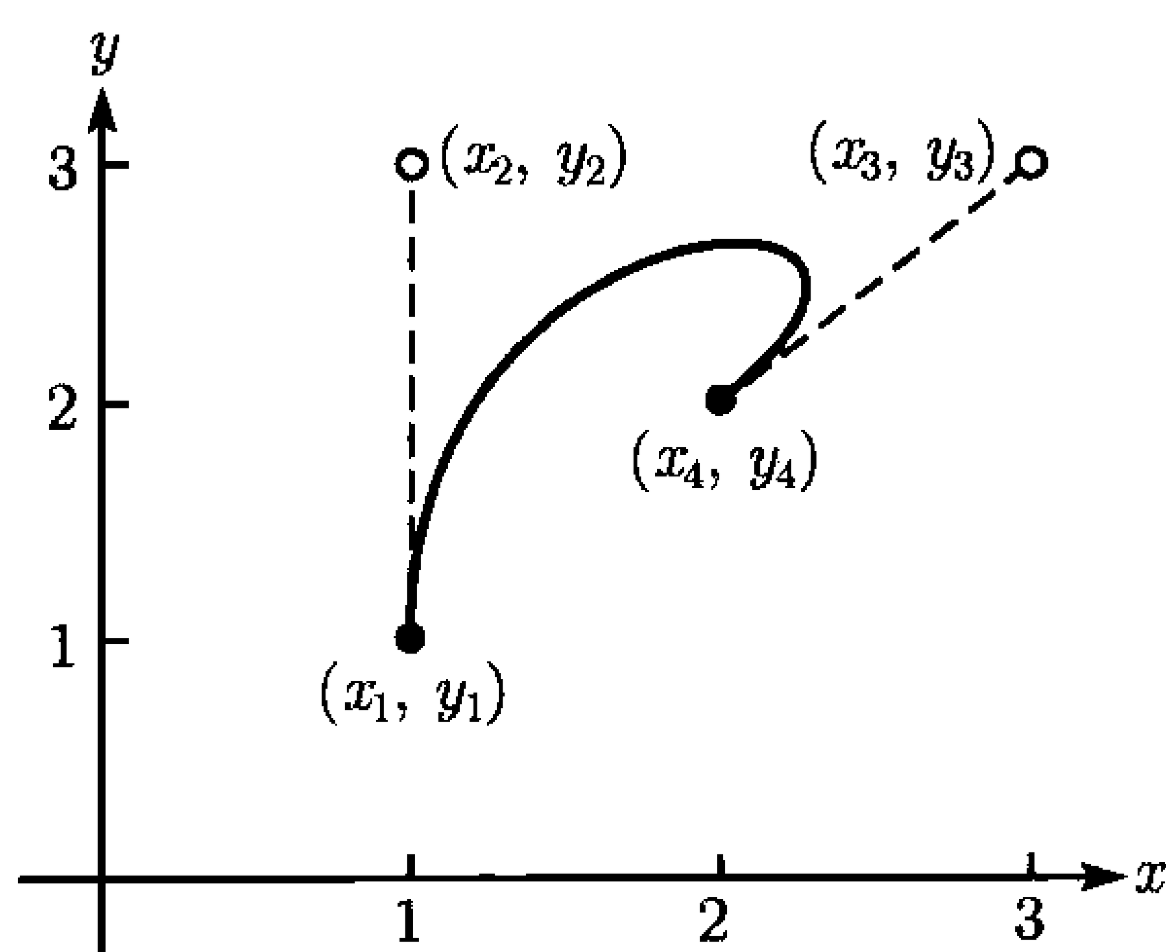


图 3-14 例 3.15 中的 Bézier 曲线. (x_1, y_1) 和 (x_4, y_4) 是样条点, 而 (x_2, y_2) 和 (x_3, y_3) 是控制点

Bézier 曲线

给定端点 (x_1, y_1), (x_4, y_4)

控制点 $(x_2, y_2), (x_3, y_3)$

令

$$
\begin{aligned}
b_x &= 3(x_2 - x_1) \\
c_x &= 3(x_3 - x_2) - b_x \\
d_x &= x_4 - x_1 - b_x - c_x \\
b_y &= 3(y_2 - y_1) \\
c_y &= 3(y_3 - y_2) - b_y \\
d_y &= y_4 - y_1 - b_y - c_y.
\end{aligned}
$$

对 $0 \leqslant t \leqslant 1$, Bézier 曲线定义为

$$
\begin{aligned}
x(t) &= x_1 + b_x t + c_x t^2 + d_x t^3 \\
y(t) &= y_1 + b_y t + c_y t^2 + d_y t.^3
\end{aligned}
$$

从方程中容易检验上一段落中的声明. 事实上, 根据习题 9,

$$\begin{aligned}
x(0) &= x_1, \\
x'(0) &= 3(x_2 - x_1), \\
x(1) &= x_4, \\
x'(1) &= 3(x_4 - x_3),
\end{aligned} \tag{3.31}$$

而且对 $y(t)$ 也有类似的事实成立.

例 3.15 求经过点 (1, 1), (2, 2) 且其控制点为 (1, 3) 和 (3, 3) 的 Bézier 曲线 $(x(t), y(t))$.

4 个点分别是 $(x_1, y_1) = (1,1)$, $(x_2, y_2) = (1,3)$, $(x_3, y_3) = (3,3)$, $(x_4, y_4) = (2,2)$. 由 Bézier 公式得到 $b_x = 0$, $c_x = 6$, $d_x = -5$, $b_y = 6$, $c_y = -6$, $d_y = 1$. Bézier 样条

$$\begin{aligned}
x(t) &= 1 + 6t^2 - 5t^3, \\
y(t) &= 1 + 6t - 6t^2 + t^3
\end{aligned}$$

与控制点一起展示在图 3-14 中. ◀

Bézier 曲线是能够搭起来拟合任何函数值和斜率的构造块 (building block). 在节点处的斜率可以按用户要求而确定, 在这种意义上, 它们是对三次样条的一种改进. 然而, 这种自由会带来光滑度的损失: 从两个不同的方向, 二阶导数在节点处一般不相等. 在某些应用中, 这种不相等反而是一种优点.

作为特殊情形, 当控制点等于端点时, 如例 3.16 所示, 样条就是简单的直线段.

例 3.16 在 $(x_1, y_1) = (x_2, y_2)$ 以及 $(x_3, y_3) = (x_4, y_4)$ 时, 证明 Bézier 样条是直线段. Bézier 公式表明, 方程是

$$\begin{aligned}
x(t) &= x_1 + 3(x_4 - x_1)t^2 - 2(x_4 - x_1)t^3 = x_1 + (x_4 - x_1)t^2(3 - 2t), \\
y(t) &= y_1 + 3(y_4 - y_1)t^2 - 2(y_4 - y_1)t^3 = y_1 + (y_4 - x_1)t^2(3 - 2t).
\end{aligned}$$

其中 $0 \leqslant t \leqslant 1$. 在样条上的每一点具有形式

$$\begin{aligned}
(x(t), y(t)) &= (x_1 + r(x_4 - x_1), y_1 + r(y_4 - y_1)) \\
&= ((1 - r)x_1 + rx_4, (1 - r)y_1 + ry_4),
\end{aligned}$$

这里 $r = t^2(3 - 2t)$. 因为 $0 \leqslant r \leqslant 1$, 所以每一点落在连接 (x_1, y_1) 和 (x_4, y_4) 的直线段上. ◀

Bézier 曲线易于编程, 而且经常用于绘图软件. 平面中的手画曲线可以看作是参数曲线 $(x(t), y(t))$, 可以用 Bézier 样条表示. 方程都是在以下的 MATLAB手工绘图软件中实现. 用户点击鼠标两次以固定平面中的一个点, 一次点击确定起始点 (x_0, y_0), 第二次点击标记沿路径预定方向的第一个控制点. 另外点击两次, 确定第二个控制点和终点, 在这两个点之间画出 Bézier 样条. 每连续三次点击鼠标会把前

一个样条点用作下一段的起点, 使曲线向前进一步延伸. MATLAB命令`ginput`用来读取鼠标位置和按键的动作. 图 3-15 显示了`draw.m`的截屏.

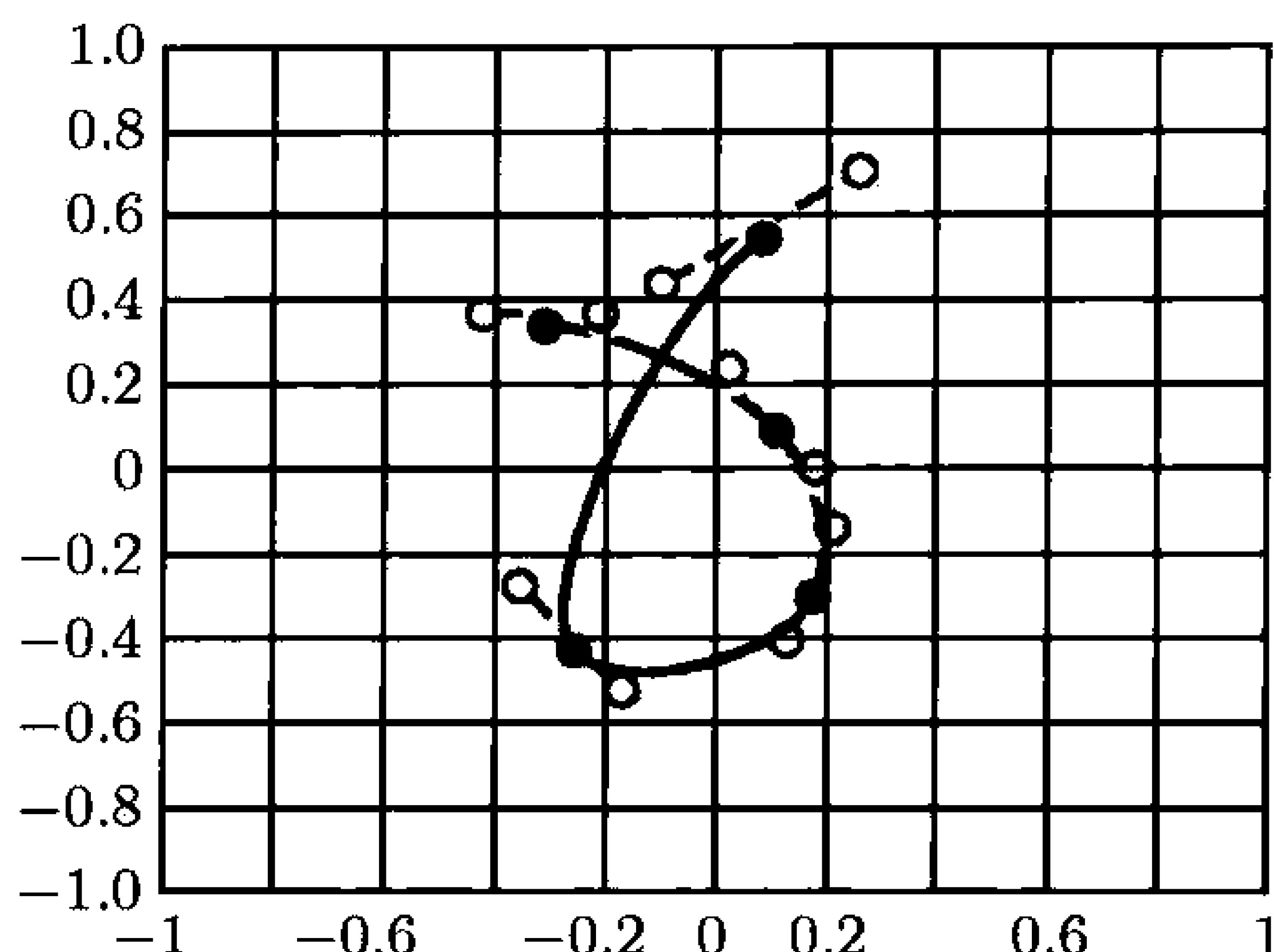


图 3-15 绘制用 Bézier 曲线编写的程序 3.7. `draw.m`绘制的屏幕图, 包括在每一个控制点的方向向量

```
%Program 3.7 Freehand Draw Program Using B\'ezier Curves
%Left click in MATLAB figure window to locate endpoint, and click
%     three more times to specify 2 control points and the next
%     spline endpoint. Continue with groups of 3 points to add
%     more to the curve. Right-click to terminate program.
function draw
hold off
plot([-1 1],[0,0],'k',[0 0],[-1 1],'k');hold on
xlist=[];ylist=[];t=0:.02:1;
button=1;k=0;
while(button ~= 3)              % if right click, terminate
  [xnew,ynew,button] = ginput(1); % get one mouse click
  if button == 1              % current click is a left click
    k=k+1;                    % k counts clicks
    xlist(k)=xnew; ylist(k)=ynew; % add new point to the list
    if k>1                    % After first input point:
      if mod(k,3) == 1        % Wait for three new input points,
        for i=1:4             % gather the previous four points
          x(i)=xlist(k-i+1);
          y(i)=ylist(k-i+1);
        end                   % Plot spline points and control pts
        plot([x(1) x(2)],[y(1) y(2)],'c:',x(2),y(2),'cs');
        plot([x(3) x(4)],[y(3) y(4)],'c:',x(3),y(3),'cs');
        plot(x(1),y(1),'bo',x(4),y(4),'bo');
        bx=3*(x(2)-x(1)); by=3*(y(2)-y(1));% spline equations ...
        cx=3*(x(3)-x(2))-bx;cy=3*(y(3)-y(2))-by;
        dx=x(4)-x(1)-bx-cx;dy=y(4)-y(1)-by-cy;
        xp=x(1)+t.*(bx+t.*(cx+t*dx));      % nested multiplication
        yp=y(1)+t.*(by+t.*(cy+t*dy));
```

```
          plot(xp,yp)          % Plot spline curve
        end
      end
    end
  end
```

虽然我们的讨论一直限制在二维 Bézier 曲线, 但是定义的方程容易推广到三维, 在三维空间中称为 Bézier 空间曲线. 就像在二维中的一样, 这种样条的每一段需要 4 个 (x, y, z) 点: 两个端点和两个控制点. 我们在习题中研究 Bézier 空间曲线的例子.

习题 3.5

1. 求由给定的 4 个点所确定的一般 Bézier 曲线 $(x(t), y(t))$:

(a) (0, 0), (0, 2) (2, 0), (1, 0); **(b)** (1, 1), (0, 0), (−2, 0), (−2, 1); **(c)** (1, 2), (1, 3), (2, 3), (2, 2).

2. 对下面的一段 Bézier 曲线求第一个端点、两个控制点和最后一个端点.

(a) $\begin{cases} x(t) = 1 + 6t^2 + 2t^3, \\ y(t) = 1 - t + t^3; \end{cases}$ **(b)** $\begin{cases} x(t) = 3 + 4t - t^2 + 2t^3, \\ y(t) = 2 - t + t^2 + 3t^3; \end{cases}$ **(c)** $\begin{cases} x(t) = 2 + t^2 - t^3, \\ y(t) = 1 - t + 2t^3. \end{cases}$

3. 求由 3 段 Bézier 曲线形成顶点为 (1, 2), (3, 4), (5, 1) 的三角形.

4. 构造形成边长为 5 的正方形的 4 段 Bézier 样条.

5. 求在端点 (−1, 0) 及 (1, 0) 处有垂直切线而且通过 (0, 1) 的一段 Bézier 样条.

6. 求在端点 (0, 1) 处有水平切线, 在端点 (1, 0) 处有垂直切线, 并且在 $t = 1/3$ 时通过 (1/3, 2/3) 的一段 Bézier 样条.

7. 求由给定的 4 点所确定的一段 Bézier 空间曲线 $(x(t), y(t), z(t))$.

(a) (1, 0, 0), (2, 0, 0), (0, 2, 1), (0, 1, 0);

(b) (1, 1, 2), (1, 2, 3), (−1, 0, 0), (1, 1, 1);

(c) (2, 1, 1), (3, 1, 1), (0, 1, 3), (3, 1, 3).

8. 求以下 Bézier空间曲线的节点及控制点.

(a) $\begin{cases} x(t) = 1 + 6t^2 + 2t^3, \\ y(t) = 1 - t + t^3, \\ z(t) = 1 + t + 6t^2; \end{cases}$ **(b)** $\begin{cases} x(t) = 3 + 4t - t^2 + 2t^3, \\ y(t) = 2 - t + t^2 + 3t^3, \\ z(t) = 3 + t + t^2 - t^3; \end{cases}$ **(c)** $\begin{cases} x(t) = 2 + t^2 - t^3, \\ y(t) = 1 - t + 2t^3, \\ z(t) = 2t^3. \end{cases}$

9. 证明 (3.13) 中的事实, 并解释它们是如何证明 Bézier 公式的.

计算机问题 3.5

1. 画出习题 5 中的曲线.

2. 画出习题 6 中的曲线.

3. 从 Bézier 曲线画出字母 (a) W, (b) B, (c) C, (d) D.

实例检验 3 Bézier 曲线中的 PostScript 字体

在这个项目中, 我们解释如何用二维 Bézier 曲线画字母和数字. 可以通过在程序 3.7 中

修改 MATLAB代码, 或者写一个 PostScript 文件来实现.

PostScript 字体直接由 Bézier 曲线建立. 下面是完整的 Post Script 文件, 演示了我们在例 3.15 中讨论的曲线.

```
%!PS
newpath
100 100 moveto
100 300 300 300 200 200 curveto
stroke
```

第一行确定文件为 PostScript 文件. **newpath**命令初始化曲线. PostScript 使用栈和后缀命令. **moveto**命令设定当前绘图点是由在栈中最后两个数字确定的 (x, y), 在这里, 是点 (100, 100). **curveto**命令从当前的栈接收 3 个 (x, y) 点, 并在当前的绘图点开始构造 Bezier 样条, 分别把这 3 个点处理为两个控制点和一个端点. **stroke**命令绘出曲线. 如果把前面的文件送到 PostScript 打印机, 图 3-14 中的 Bézier 曲线将被打印出来. 坐标已乘以 100 以便和 PostScript 的约定相匹配, 它是 72 单位对应 1 英寸. 一张信纸大小的纸是 612 PostScript 单位宽和 792 单位高.

利用**curveto**命令的 Bézier 曲线, 在计算机荧屏和打印机上绘出 PostScript 字体. 如果图形是封闭的, **fill**命令将填充轮廓线.

例如, 在 Times-Roman 字体中, 大写的 T 字母是由以下 16 条 Bézier 曲线 (每条线包括定义一段 Bézier 样条的数字 $x_1\, y_1\, x_2\, y_2\, x_3\, y_3\, x_4\, y_4$) 构造出来的:

```
237 620 237 620 237 120 237 120;
237 120 237  35 226  24 143  19;
143  19 143  19 143   0 143   0;
143   0 143   0 435   0 435   0;
435   0 435   0 435  19 435  19;
435  19 353  23 339  36 339 109;
339 109 339 108 339 620 339 620;
339 620 339 620 393 620 393 620;
393 620 507 620 529 602 552 492;
552 492 552 492 576 492 576 492;
576 492 576 492 570 662 570 662;
570 662 570 662   6 662   6 662;
  6 662   6 662   0 492   0 492;
  0 492   0 492  24 492  24 492;
 24 492  48 602  71 620 183 620;
183 620 183 620 237 620 237 620;
```

要编写一个绘制字母 T 的 PostScript 文件, 我们需要**moveto**初始点 (237, 620), 加上下面两个控制点以及端点 (237, 120), 接着用**curveto**命令. 如此之后, 当前位置将是 (237, 120), 而第二条曲线是通过命令

```
237  35 226  24 143  19 curveto
```

画出的. 另外 14 个**curveto**命令完成了字母 T, 如图 3-16 所示.

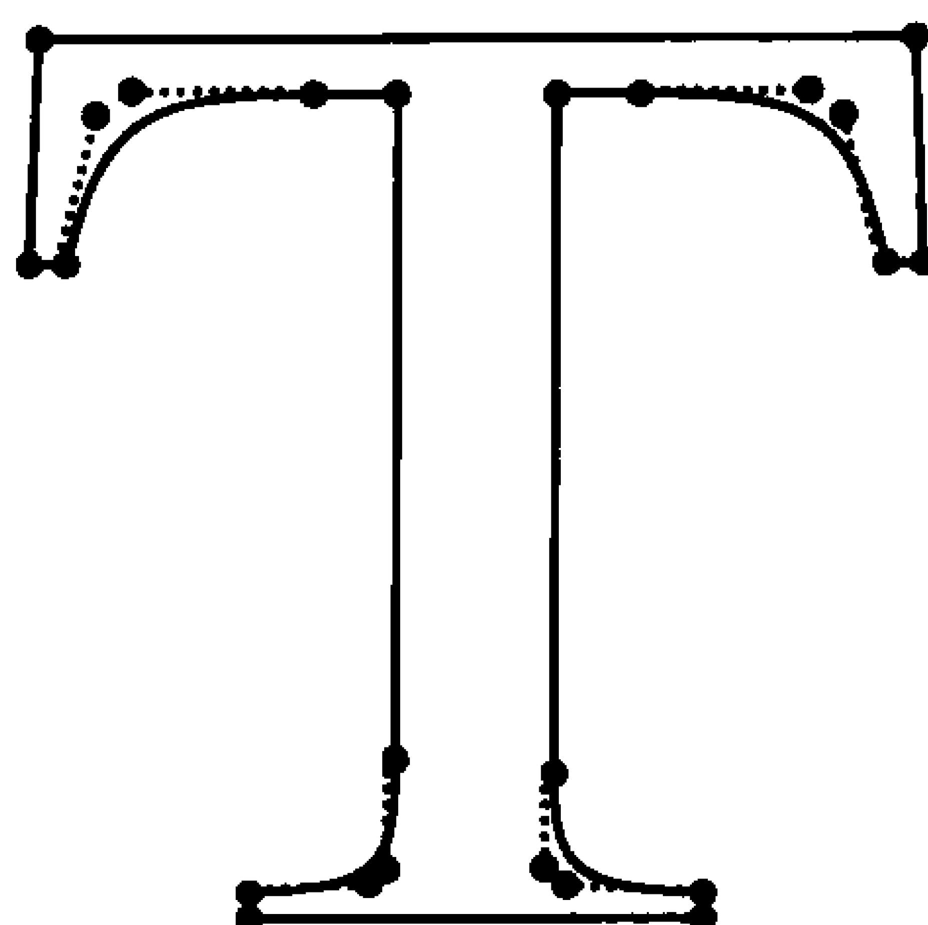


图 3-16 用 Bézier 样条绘制的 Times-Roman 字体 T. 蓝色圆点是样条端点, 黑色圆点是控制点

用以下 21 段 Bézier 曲线画出数字 5, 见图 3-17.

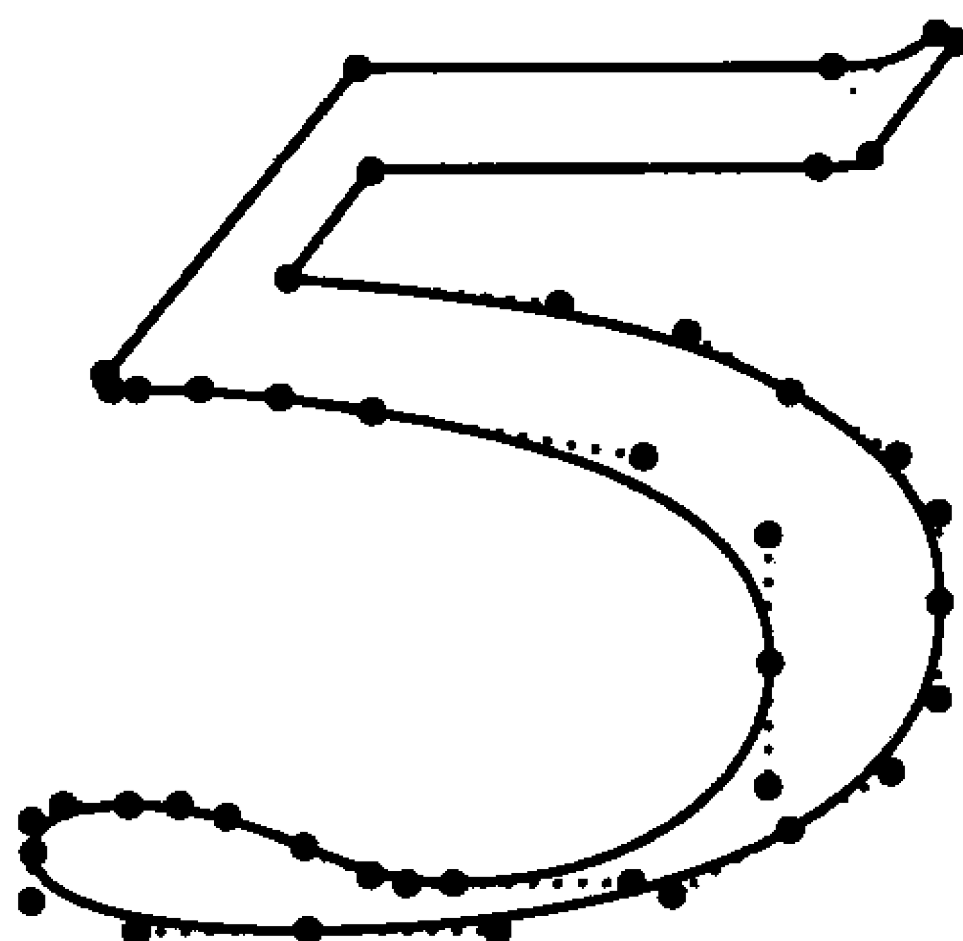


图 3-17 用 Bézier 样条制作的 Times-Roman 5: 蓝色圆是样条端点

```
149 597 149 597 149 597 345 597;
345 597 361 597 365 599 368 606;
368 606 406 695 368 606 406 695;
406 695 397 702 406 695 397 702;
397 702 382 681 372 676 351 676;
351 676 351 676 351 676 142 676;
142 676  33 439 142 676  33 439;
 33 439  32 438  32 436  32 434;
 32 434  32 428  35 426  44 426;
 44 426  74 426 109 420 149 408;
149 408 269 372 324 310 324 208;
324 208 324 112 264  37 185  37;
185  37 165  37 149  44 119  66;
119  66  86  90  65  99  42  99;
 42  99  14  99   0  87   0  62;
  0  62   0  24  46   0 121   0;
121   0 205   0 282  27 333  78;
333  78 378 123 399 180 399 256;
399 256 399 327 381 372 333 422;
```

```
333 422 288 468 232 491 112 512;
112 512 112 512 149 597 149 597;
```

建议习题

1. 用 3.5 节中的`draw.m`程序, 画出你的姓中第一个字母的大写形式.

2. 修改`draw`程序以接受 $n \times 8$ 数字矩阵, 每一行表示一段 Bézier 样条. 用以下 21 段 Bézier 样条, 让程序画出 Times-Roman 字体的字母`f`.

```
289 452 289 452 166 452 166 452;
166 452 166 452 166 568 166 568;
166 568 166 627 185 657 223 657;
223 657 245 657 258 647 276 618;
276 618 292 589 304 580 321 580;
321 580 345 580 363 598 363 621;
363 621 363 657 319 683 259 683;
259 683 196 683 144 656 118 611;
118 611  92 566  84 530  83 450;
 83 450  83 450   1 450   1 450;
  1 450   1 450   1 418   1 418;
  1 418   1 418  83 418  83 418;
 83 418  83 418  83 104  83 104;
 83 104  83  31  72  19   0  15;
  0  15   0  15   0   0   0   0;
  0   0   0   0 260   0 260   0;
260   0 260   0 260  15 260  15;
260  15 178  18 167  29 167 104;
167 104 167 104 167 418 167 418;
167 418 167 418 289 418 289 418;
289 418 289 418 289 452 289 452;
```

3. 编写 PostScript 文件来画字母`f`. 程序应该用`moveto`来确定初始点, 随后是 21 条`curveto`命令. 通过在 Ghostscript 视窗打开你的文件或者把它送到 PostScript 打印机来测试这个程序.

4. 尽管字体信息多年来是严密保护的秘密, 但是现在其中许多可在 Web 上自由获得. 搜索其他的字体, 并找出用 PostScript 或者用`draw.m`绘出你所选择的字母的 Bézier 曲线.

5. 设计你自己的字母或数字. 先在图纸上画出图形, 表示出可能会有的对称性. 估计控制点, 并准备今后在需要时修改它们.

软件和进一步阅读

插值软件通常由确定和计算插值多项式的独立代码组成. 为此目的, MATLAB提供了`Polyfit`和`Polyval`命令. MATLAB的`spline`命令默认情况下计算非节点样条, 但是也可以选择其他命令的端点条件. 命令`interp1`把几种一维插值的选择结合起来. NAG 库包括了用于多项式和样条插值的子程序`e01aef`和`e01baf`, IMSL 库有许多基于各种端点条件的样条程序.

关于插值基本知识的经典参考书见 [2], 而参考文献 [5, 6] 覆盖了函数近似和 Chebyshev 插值. DeBoor 关于样条的教科书 [3] 也很经典, 另见 [7] 及 [8]. 关于计算机辅助建模和设计的应用, 请参见 [4] 和 [10]. 关于近似特殊函数的 CORDIC 方法在 [9] 中介绍.

第4章 最小二乘

全球定位系统 (GPS) 是一种基于卫星的定位技术, 它可随时对地球上的任何一点提供精确的位置. 飞行员、船长和司机应用 GPS 来避免碰撞和准确定位, 农民用它来控制通过田地的器械, 徒步旅行者和驾船的人把它当作导航的工具.

这个系统包括 24 颗卫星, 它们沿着精确控制的轨道, 并发射同步信号. 地基接收机接收这些卫星信号, 求出它与所有可见的卫星的距离, 并用这些数据通过三角测量确定它的位置.

实例检验 4.4 节后的实例检验表示应用方程解法和最小二乘计算来进行位置估计.

最小二乘的概念是从 19 世纪早期 Gauss 和 Legendre 的开拓性工作开始的. 它的应用遍及现代统计学和数学建模. 这种回归和参数估计的关键技术在科学和工程中已经成为基本工具.

在本章里引入正规方程并将其应用到各种数据拟合问题中. 后面, 会研究一种应用 QR 分解的更复杂的方法, 然后讨论非线性最小二乘问题.

4.1 最小二乘和正规方程

对于最小二乘方法的需要分别来自我们在第 2 章和第 3 章研究的内容. 在第 2 章中, 我们学习了当解存在时, 如何求 $Ax = b$ 的解. 在本章我们要学习当解不存在时, 我们该如何做. 当方程组不相容时, 或许是方程的个数多于未知量的个数, 那么答案是求次佳 (next-best) 解 —— 最小二乘近似.

第 3 章谈到寻求准确拟合数据点的多项式. 然而, 如果有大量的数据点或者数据点仅是在某种误差范围内收集到的, 那么准确地拟合高次多项式一般不是最佳的方法. 这时拟合一个可能仅是近似数据点的更简单模型更为合理. 解不相容方程组和近似地拟合数据都推动了最小二乘法的发展.

4.1.1 不相容方程组

不难写出没有解的方程组. 考虑下面含两个未知量的 3 个方程

$$\begin{cases} x_1 + x_2 = 2, \\ x_1 - x_2 = 1, \\ x_1 + x_2 = 3. \end{cases} \tag{4.1}$$

任一解必须满足第一个和第三个方程, 而这两个方程不可能都是正确的. 一个无解的方程组称为**不相容**.

方程组无解意味着什么? 或许是系数稍不精确. 在许多情形下, 方程的个数大于未知量的个数, 使解不大可能满足所有的方程. 事实上, 当 $m > n$ 时, 含 n 个未知量的 m 个方程一般无解. 尽管 Gauss 消去法对于不相容方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 将不会给出解, 我们也不应该完全放弃. 在这种情形下可采用的方法是求一个最接近于解的向量 $\boldsymbol{x}$.

如果我们这个 "接近" 是欧氏距离下的接近, 那么就有一种求最接近的 $\boldsymbol{x}$ 的直接算法. 我们把这种特别的 $\boldsymbol{x}$ 称为**最小二乘解**(least squares solution).

通过用不同的方式写出方程组 (4.1), 我们可以得到原本没有解的方程组更好的结果, 并求出解. 方程组的矩阵形式是 $\boldsymbol{Ax} = \boldsymbol{b}$, 或者

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}. \tag{4.2}$$

把矩阵/向量乘法写成如下的等价方程

$$x_1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}. \tag{4.3}$$

事实上, 任一 $m \times n$ 方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 都可以看作向量方程

$$x_1\boldsymbol{v}_1 + x_2\boldsymbol{v}_2 + \cdots + x_n\boldsymbol{v}_n = \boldsymbol{b}, \tag{4.4}$$

它把 $\boldsymbol{b}$ 表示成 $\boldsymbol{A}$ 的列向量的线性组合, 其系数是 $x_1, \cdots, x_n$. 因为两个线性无关向量的线性组合形成 $\mathbb{R}^3$ 内的一个平面, 所以仅当向量 $\boldsymbol{b}$ 落在这个平面内, 方程 (4.3) 有解. 当我们试图解含 n 个未知量的 m 个方程时 $(m > n)$, 情形将永远是这样. 太多的方程使问题超定, 并且方程不相容.

当解不存在时, 图 4-1b 为我们指出了一个方向. 不存在解 (4.1) 的一组 x_1, x_2, 但是, 在一切可能的候选解中, 平面 $\boldsymbol{Ax}$ 中存在一点, 它与 $\boldsymbol{b}$ 最为接近. 这一特殊向量 $\boldsymbol{A}\bar{\boldsymbol{x}}$ 的特别之处在于以下事实: 差向量 $\boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}}$ 垂直于平面 $\{\boldsymbol{Ax} | \boldsymbol{x} \in \mathbb{R}^n\}$. 我们将用这一事实来寻求最小二乘 **"解"** $\bar{\boldsymbol{x}}$ 的公式.

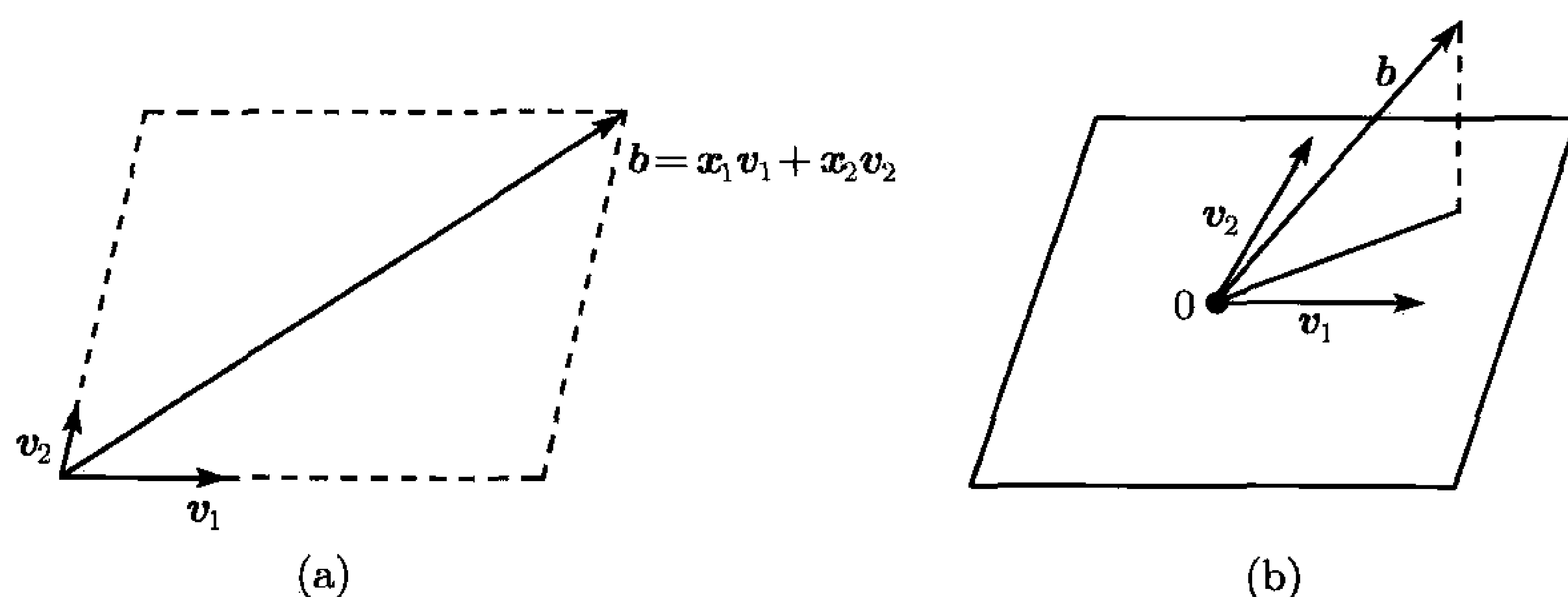


图 4-1 含两个未知量的 3 个方程的方程组的几何解. (a) 方程 (4.3) 要求方程的右端向量 $\boldsymbol{b}$ 是列向量 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 的向量组合. (b) 如果 $\boldsymbol{b}$ 落在由 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 所确定的平面之外, 将无解. 最小二乘解 $\bar{\boldsymbol{x}}$ 使得向量组合 $\boldsymbol{A}\bar{\boldsymbol{x}}$ 在平面 $\boldsymbol{A}\boldsymbol{x}$ 中, 在欧氏距离的意义下它与 $\boldsymbol{b}$ 最为接近

首先, 我们创建一些记号. 回忆一下 $m \times n$ 矩阵 $\boldsymbol{A}$ 的**转置**$\boldsymbol{A}^{\mathrm{T}}$ 的概念, 它是 $m \times n$ 矩阵, 其行为 $\boldsymbol{A}$ 的列, 并且其列为 $\boldsymbol{A}$ 的行, 次序相同. 两个矩阵的和的转置是转置的和, $(\boldsymbol{A}+\boldsymbol{B})^{\mathrm{T}} = \boldsymbol{A}^{\mathrm{T}} + \boldsymbol{B}^{\mathrm{T}}$. 两个矩阵的积转置是转置逆序的积, 即 $(\boldsymbol{A}\boldsymbol{B})^{\mathrm{T}} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}$.

为了利用正交性, 回忆一下, 如果两个向量的内积是零, 那么这两个向量互成直角. 对于两个 m 维向量 $\boldsymbol{u}$ 及 $\boldsymbol{v}$, 我们可以把内积写成矩阵乘法的形式:

$$\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} = [u_1, \cdots, u_m]\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}. \tag{4.5}$$

利用通常的矩阵乘法, 如果 $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} = 0$, 那么向量 $\boldsymbol{u}$ 与 $\boldsymbol{v}$ 正交.

亮点 正交性

最小二乘是以正交性为基础的. 一个点到一个平面的最短距离通过正交于平面的直线段获得. 正规方程是确定直线段位置的计算方法, 它表示最小二乘误差.

现在, 我们回来寻求 $\bar{\boldsymbol{x}}$ 的公式. 我们已经确立

$$(\boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}}) \perp \{\boldsymbol{A}\boldsymbol{x} | \boldsymbol{x} \in \mathbb{R}^n\}.$$

用矩阵乘法来表示正交性, 我们得到

$$(\boldsymbol{A}\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}}) = 0, \quad x \in \mathbb{R}^n.$$

利用前面关于转置的性质, 可以把这一表达式重新写成

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\bar{\boldsymbol{x}})=0,\quad x\in\mathbb{R}^n,$$

这就意味着 n 维向量 $\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\bar{\boldsymbol{x}})$ 正交于 $\mathbb{R}^n$ 中的每一个向量 $\boldsymbol{x}$, 并且包括它自己. 这只可能是

$$\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\bar{\boldsymbol{x}})=0.$$

它给出了确定最小二乘解的方程组,

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\bar{\boldsymbol{x}}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}. \tag{4.6}$$

方程组 (4.6) 称为**正规方程**. 它的解就是方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 的所谓的最小二乘解.

最小二乘的正规方程

给定不相容方程组

$$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b},$$

解

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\bar{\boldsymbol{x}}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$$

得到最小二乘解 $\bar{\boldsymbol{x}}$.

例 4.1 用正规方程求不相容方程组 (4.1) 的最小二乘解.

在问题的矩阵形式 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 中,

$$\boldsymbol{A}=\begin{bmatrix}1&1\\1&-1\\1&1\end{bmatrix},\qquad \boldsymbol{b}=\begin{bmatrix}2\\1\\3\end{bmatrix}.$$

正规方程的组成部分是

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}=\begin{bmatrix}1&1&1\\1&-1&1\end{bmatrix}\begin{bmatrix}1&1\\1&-1\\1&1\end{bmatrix}=\begin{bmatrix}3&1\\1&3\end{bmatrix}$$

及

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}=\begin{bmatrix}1&1&1\\1&-1&1\end{bmatrix}\begin{bmatrix}2\\1\\3\end{bmatrix}=\begin{bmatrix}6\\4\end{bmatrix}.$$

正规方程

$$\begin{bmatrix}3&1\\1&3\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}6\\4\end{bmatrix}$$

可以用 Gauss 消去法解出. 表格形式是

$$\left[\begin{array}{cc|c} 3 & 1 & 6 \\ 1 & 3 & 4 \end{array}\right] \to \left[\begin{array}{cc|c} 3 & 1 & 6 \\ 0 & 8/3 & 2 \end{array}\right],$$

我们可以解得 $\bar{\boldsymbol{x}} = (\bar{x}_1, \bar{x}_2) = \left(\frac{7}{4}, \frac{3}{4}\right)$.

把最小二乘解代入原问题得到

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{7}{4} \\ \frac{3}{4} \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1 \\ 2.5 \end{bmatrix} \neq \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}.$$

为了估计我们在数据拟合上的成果, 计算最小二乘解 $\bar{\boldsymbol{x}}$ 的残差:

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 2.5 \\ 1 \\ 2.5 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.0 \\ 0.5 \end{bmatrix}.$$

如果残差是零向量, 那么我们已经精确地解出原方程组 $\boldsymbol{Ax} = \boldsymbol{b}$. 否则, 残差向量的欧氏长度就是 $\bar{\boldsymbol{x}}$ 离真解 $\boldsymbol{x}$ 多远的一种度量. 向量的欧氏长度

$$\|\boldsymbol{r}\|_2 = \sqrt{r_1^2 + \cdots + r_m^2}, \tag{4.7}$$

是一种范数, 称为2- **范数**. **平方误差**(squared error).

$$\mathrm{SE} = r_1^2 + \cdots + r_m^2$$

以及**均方根误差**(root mean squared error)

$$\mathrm{RMSE} = \sqrt{\frac{\mathrm{SE}}{m}} = \sqrt{(r_1^2 + \cdots + r_m^2)/m}, \tag{4.8}$$

也就是平方误差的平均值的平方根, 也用来测量最小二乘解的误差. 对于例 4.1, $\mathrm{SE} = (0.5)^2 + 0^2 + (-0.5)^2 = 0.5$, 误差的 2- 范数是 $\|\boldsymbol{r}\|_2 = \sqrt{0.5} \approx 0.707$, 并且

$$\mathrm{RMSE} = \sqrt{\frac{0.5}{3}} = \frac{1}{\sqrt{6}} \approx 0.408.$$

例 4.2 解最小二乘问题

$$\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix}.$$

正规方程 $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$ 是

$$\begin{bmatrix} 9 & 6 \\ 6 & 29 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 45 \\ 75 \end{bmatrix}.$$

正规方程的解是 $\bar{x}_1 = 3.8, \bar{x}_2 = 1.8$. 残差向量是

$$\begin{aligned} r = b - A\bar{x} &= \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 3.8 \\ 1.8 \end{bmatrix} \\ &= \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix} - \begin{bmatrix} -3.4 \\ 13 \\ 11.2 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 2 \\ -2.2 \end{bmatrix}, \end{aligned}$$

它有欧氏范数

$$\|r\|_2 = \sqrt{(0.4)^2 + 2^2 + (-2.2)^2} = 3.$$

例 4.14 中用另一种方法求解了这个问题.

4.1.2 数据拟合模型

设 $(t_1, y_1), \cdots, (t_m, y_m)$ 是平面中的一组点集, 我们将经常称它们为 “数据点”. 给出一类确定的模型, 譬如所有的直线 $y = a + bt$, 我们可以寻找在 2- 范数意义下最佳拟合这些数据点的这种模型的特定实例. 最小二乘思想的核心包括: 通过模型在数据点的平方误差来测量拟合的残差, 以及寻求使残差极小化的模型参数. 图 4-2 显示了这一准则.

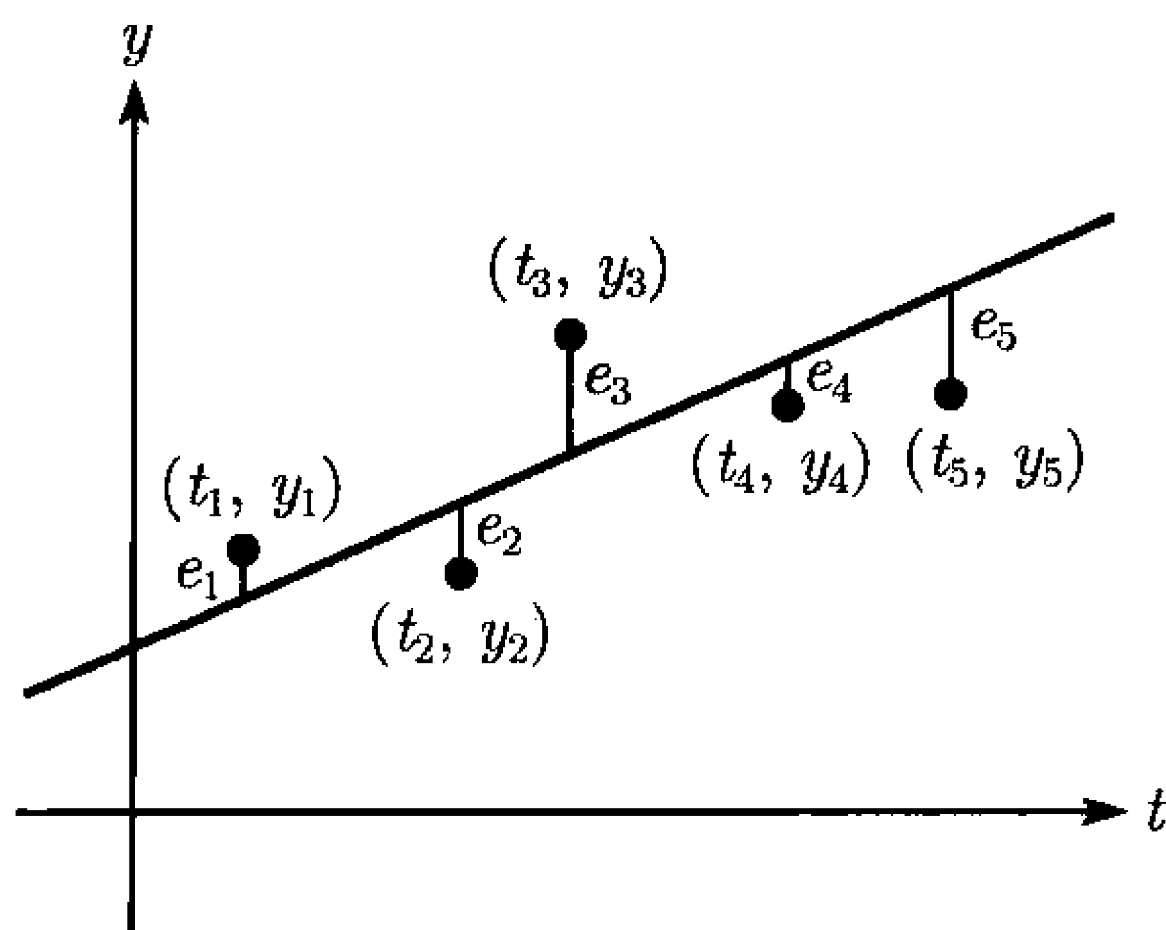


图 4-2 直线对数据的最小二乘拟合: 最佳直线是在所有直线 $y = a + bt$ 中平方误差 $e_1^2 + e_2^2 + \cdots + e_5^2$ 尽可能小的那一条

例 4.3 求最佳拟合图 4-3 中 3 个数据点 $(t, y) = (1, 2), (-1, 1), (1, 3)$ 的直线.

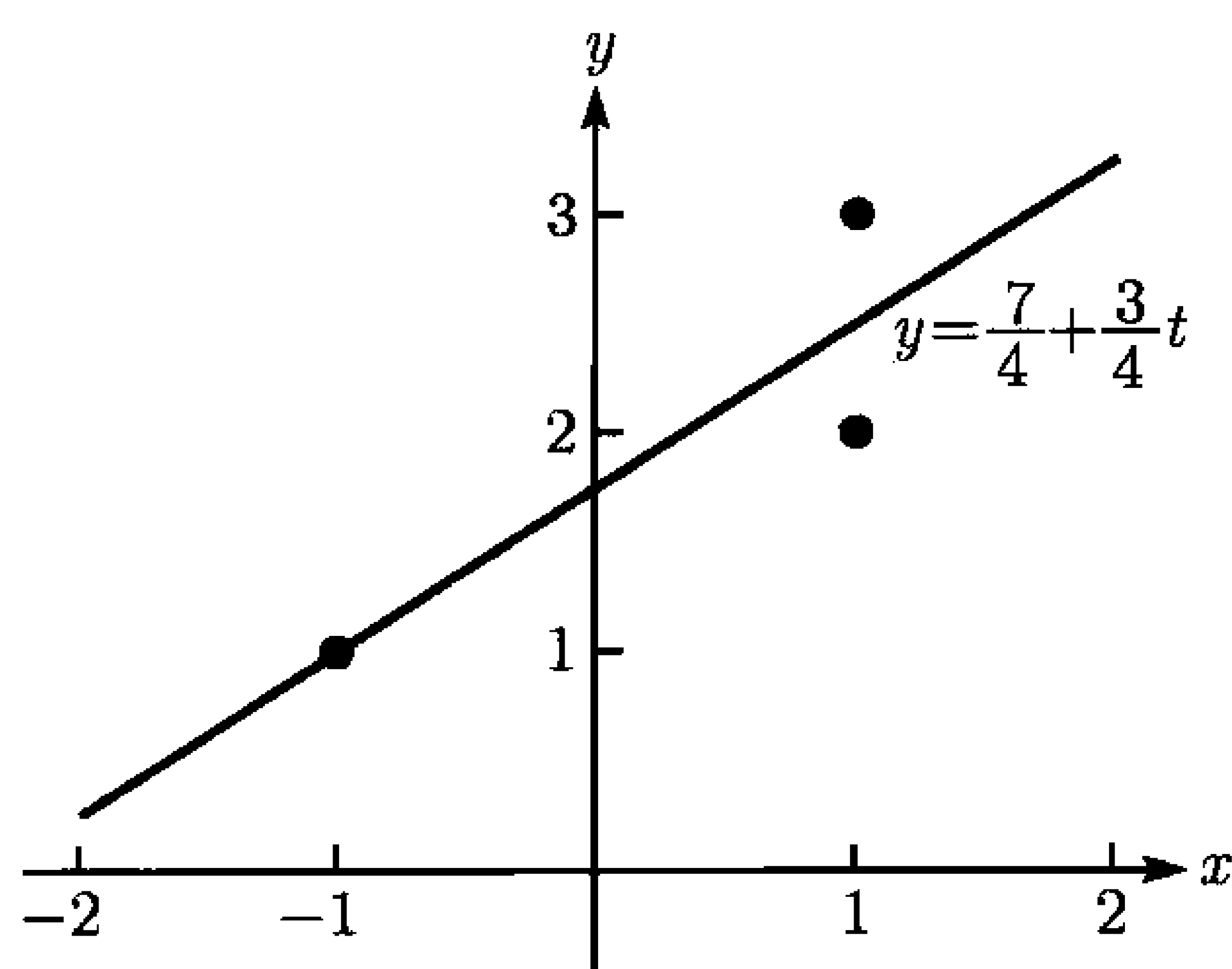


图 4-3 例 4.3 中的最佳直线：数据点分别落在最佳直线的上方及下方

模型是 $y = a + bt$, 目标是求最佳的 a 及 b. 把数据点代入模型得到

$$
\begin{aligned}
a + b(1) &= 2, \\
a + b(-1) &= 1, \\
a + b(1) &= 3.
\end{aligned}
$$

或者写成矩阵形式

$$
\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}.
$$

我们知道这个方程组不存在解 (a, b), 原因如下. 首先, 如果存在解, 那么 $y = a + bt$ 应是一条包含这 3 个数据点的直线. 然而易见这 3 点不共线. 其次, 它是在本章开始时讨论过的方程组 (4.2). 然后, 注意到第一个方程和第三个方程不相容, 就最小二乘而言, 我们求得的最佳解是 $(a, b) = \left(\frac{7}{4}, \frac{3}{4}\right)$. 因此最佳直线是 $y = \frac{7}{4} + \frac{3}{4}t$. ◀

我们用以前定义的统计量来估计这种拟合. 在数据点的残差是

t	y	直线	误差
1	2	2.5	−0.5
−1	1	1.0	0.0
1	3	2.5	0.5

如前所见,RMSE 是 $\frac{1}{\sqrt{6}}$.

前面的例子对求解最小二乘数据拟合问题提出了 3 步程序.

最小二乘数据拟合

给定一组 m 个数据点 $(t_1, y_1), \cdots, (t_m, y_m)$.

第一步：选取模型. 确定参数化模型, 如 $y = a + bt$, 并将用它来拟合数据.

第二步：使模型拟合数据. 把数据点代入模型. 每个数据点产生一个将未知量作为参数的方程, 譬如在直线模型中的 a 与 b. 结果产生方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$, 在这里未知量 $\boldsymbol{x}$ 表示未知参数.

第三步：求解正规方程. 参数的最小二乘解将作为正规方程组 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ 的解而求得.

亮点　压缩

最小二乘是数据压缩的典型例子. 输入包括一组数据点. 而输出是尽可能好地拟合数据的模型, 它带有相对少的参数. 通常使用最小二乘的原因是用合理的模型来代替噪声数据. 于是, 为了信号预测和分类的目的, 经常使用这种模型.

在 4.4 节, 各种不同的模型被用于拟合数据, 包括多项式、指数函数以及三角函数. 在第 10 章及第 11 章将进一步应用三角逼近, 介绍基本的 Fourier 分析作为信号处理的初步.

我们在下例中说明这些步骤.

例 4.4 对图 4-4 中的 4 个数据点 $(-1,1),(0,0),(1,0),(2,-2)$, 求最佳直线和最佳抛物线.

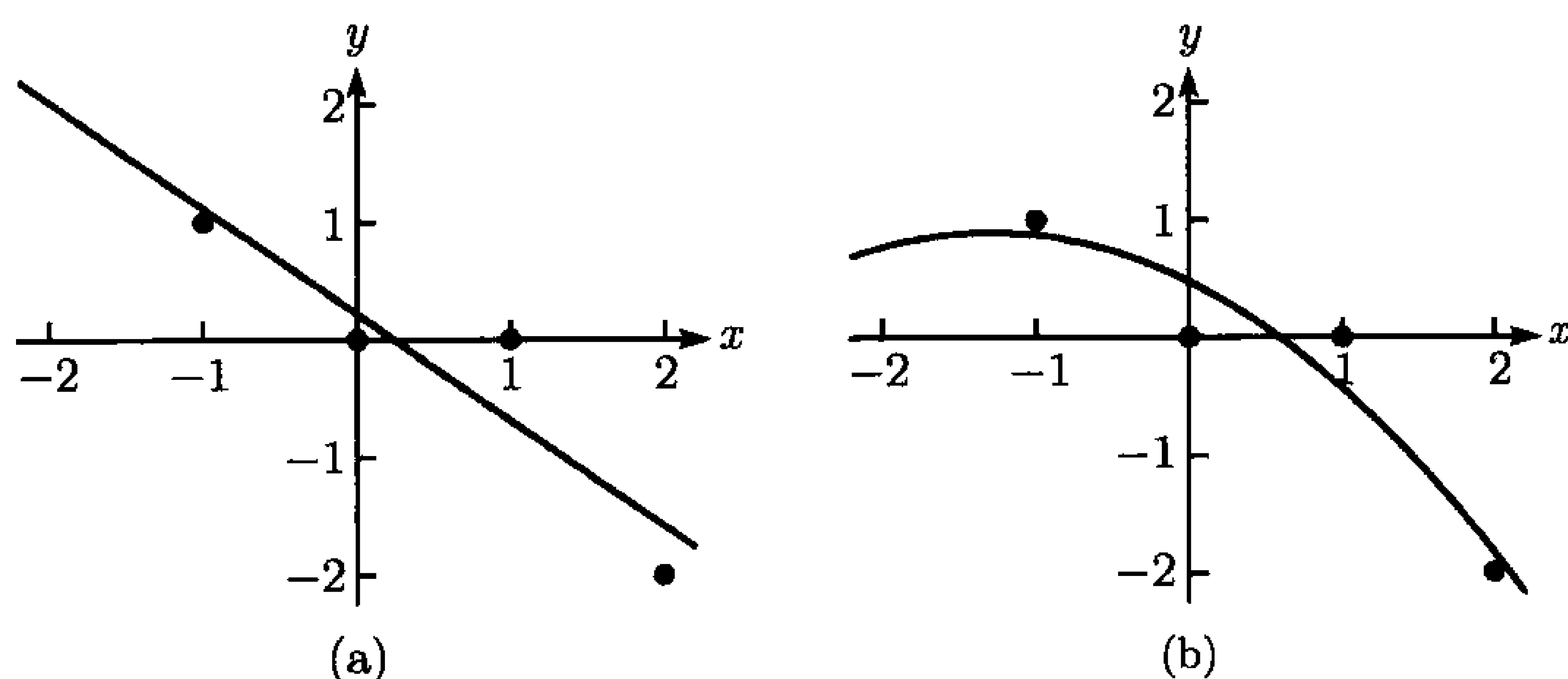


图 4-4　对例 4.4 中的数据点的最小二乘拟合：(a) 最佳直线 $y=0.2-0.9t$, RMSE=0.418; (b) 最佳抛物线 $y=0.45-0.65t-0.25t^2$, RMSE=0.335

按照前面的程序, 我们将采用 3 步：首先像前面一样选取模型 $y=a+bt$, 其次使模型拟合数据得到

$$\begin{cases} a+b(-1)=1, \\ a+b(0)=0, \\ a+b(1)=0, \\ a+b(2)=-2, \end{cases}$$

或者写成矩阵形式

$$\begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \end{bmatrix}.$$

正规方程是

$$\begin{bmatrix} 4 & 2 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -1 \\ -5 \end{bmatrix}.$$

解出系数 a 与 b, 结果得到最佳直线 $y = a + bt = 0.2 - 0.9t$.

残差是

t	y	直线	误差
−1	1	1.1	−0.1
0	0	0.2	−0.2
1	0	−0.7	0.7
2	−2	−1.6	−0.4

误差统计量是平方误差 $\mathrm{SE} = (-0.1)^2 + (-0.2)^2 + (0.7)^2 + (-0.4)^2 = 0.7$ 以及 $\mathrm{RMSE} = \sqrt{0.7}/\sqrt{4} = 0.418$.

下面我们保持同样的 4 个数据点, 但是改变模型来扩展这个例子. 设 $y = a + bt + ct^2$, 并且代入数据点得到

$$\begin{cases} a + b(-1) + c(-1)^2 = 1, \\ a + b(0) + c(0)^2 = 0, \\ a + b(1) + c(1)^2 = 0, \\ a + b(2) + c(2)^2 = -2, \end{cases}$$

或者写成矩阵形式

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -2 \end{bmatrix}.$$

这一次, 正规方程是含 3 个未知量的 3 个方程:

$$\begin{bmatrix} 4 & 2 & 6 \\ 2 & 6 & 8 \\ 6 & 8 & 18 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ -5 \\ -7 \end{bmatrix}.$$

解出系数, 结果得到最佳抛物线 $y=a+bt+ct^2=0.45-0.65t-0.25t^2$. 下表给出残差.

t	y	抛物线	误差
−1	1	0.85	0.15
0	0	0.45	−0.45
1	0	−0.45	0.45
2	−2	−1.85	−0.15

误差统计量是平方误差 $\mathrm{SE}=(0.15)^2+(-0.45)^2+(0.45)^2+(-0.15)^2=0.45$ 以及 $\mathrm{RMSE}=\sqrt{0.45}/\sqrt{4}\approx 0.335$.

亮点　条件作用

由于在最小二乘问题中假定输入数据有误差, 因此减小误差的放大特别重要. 我们已提出正规方程作为解最小二乘问题的最直接方法, 而且它适用于小型问题. 然而条件数 $\mathrm{cond}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})$ 近似于原来的条件数 $\mathrm{cond}(\boldsymbol{A})$ 的平方, 这将极大地增加问题是病态的可能性. 更复杂的方法允许不必形成 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 而直接从 $\boldsymbol{A}$ 计算最小二乘解. 这些方法建立在 4.3 节介绍的 $\boldsymbol{QR}$ 分解以及第 12 章的奇异值分解的基础上.

MATLAB命令`polyfit`及`polyval` 不但是对多项式模型的插值数据设计的, 而且也是对拟合数据设计的. 对于 n 个数据点, 使用 $n-1$ 次输入数据的`polyfit` 返回 $n-1$ 次插值多项式的系数. 如果输入次数小于 $n-1$, `polyfit` 代之以寻求该次数的最佳最小二乘多项式. 例如, 命令

```
>> x0=[-1 0 1 2];
>> y0=[1 0 0 -2];
>> c=polyfit(x0,y0,2);
>> x=-1:.01:2;
>> y=polyval(c,x);
>> plot(x0,y0,'o',x,y)
```

求二次最小二乘多项式的系数, 并且利用例 4.4 给出的数据把它画出来.

例 4.4 说明最小二乘模型不需要被限制为寻求最佳直线. 通过推广模型的定义, 我们可以对任一模型拟合系数, 只要这些系数是以线性方式进入模型的.

4.1.3 最小二乘的条件作用

我们已经看到, 最小二乘问题简化成求解正规方程 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\bar{x}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$. 可以确定最小二乘解 $\bar{x}$ 的精度吗? 这是有关正规方程的前向误差问题. 在准确解知道的情形下通过解正规方程, 我们进行双精度数值试验来测试这个问题.

例 4.5　设 $x_1=2.0, x_2=2.2, x_3=2.4, \cdots, x_{11}=4.0$ 是在 $[2,4]$ 中的等距点,

并且令 $y_i = 1 + x_i + x_i^2 + x_i^3 + x_i^4 + x_i^5 + x_i^6 + x_i^7 (1 \leqslant i \leqslant 11)$. 利用正规方程求拟合 (x_i, y_i) 的最小二乘多项式 $P(x) = c_1 + c_2 x + \cdots + c_8 x^7$.

7 次多项式用于拟合落在 7 次多项式 $P(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$ 上的 11 个数据点. 显然正确的最小二乘解是 $c_1 = c_2 = \cdots = c_8 = 1$. 把数据点代入模型 $P(x)$, 得到方程组 $\boldsymbol{Ac} = \boldsymbol{b}$:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^7 \\ 1 & x_2 & x_2^2 & \cdots & x_2^7 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{11} & x_{11}^2 & \cdots & x_{11}^7 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_8 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{11} \end{bmatrix}.$$

系数矩阵是**Van der Monde 矩阵**, 这个矩阵的第 j 列元素是由第 2 列元素的 $(j-1)$ 次幂组成. 我们用 MATLAB来解正规方程:

```
>> x = (2+(0:10)/5)';
>> y = 1+x+x.^2+x.^3+x.^4+x.^5+x.^6+x.^7;
>> A = [x.^0 x x.^2 x.^3 x.^4 x.^5 x.^6 x.^7];
>> c = (A'*A)\(A'*y)

c=
    3.8938
   -6.1483
    8.4914
   -3.3182
    2.4788
    0.6990
    1.0337
    0.9984

>> cond(A'*A)

ans=
  1.3674e+019
```

用双精度解正规方程并不能提供最小二乘解的精确值. $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 的条件数太大以至无法处理双精度算法, 并且即使原来的最小二乘问题的条件数适中, 正规方程也是病态的. 用正规方程逼近最小二乘, 显然有待改进. 在例 4.15 中, 选择了另一种方案, 避免形成 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, 到时我们再回到这个问题. ◀

习题 4.1

1. 对以下的不相容方程组, 解正规方程求最小二乘解以及 2- 范数误差:

(a) $\begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$; **(b)** $\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$;

(c) $\begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 2 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 2 \end{bmatrix}$.

2. 求以下方程组的最小二乘解以及 RMSE:

(a) $\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 3 \\ 4 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 2 \\ 1 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 2 \end{bmatrix}$.

3. 求不相容方程组的最小二乘解:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 6 \end{bmatrix}.$$

4. 设 $m \geqslant n$, $\boldsymbol{A}$ 是 $m \times n$ 单位矩阵 ($m \times m$ 单位矩阵的主子矩阵), 并且令 $\boldsymbol{b} = [b_1, \cdots, b_m]$ 是向量. 求 $\boldsymbol{Ax} = \boldsymbol{b}$ 的最小二乘解及 2- 范数误差.

5. 证明 2- 范数是一种向量范数. 你将需用 Cauchy-Schwarz 不等式 $|\boldsymbol{u} \cdot \boldsymbol{v}| \leqslant \|\boldsymbol{u}\|_2 \|\boldsymbol{v}\|_2$.

6. 设 $\boldsymbol{A}$ 是 $n \times n$ 非奇异矩阵. **(a)** 证明 $(\boldsymbol{A}^{\mathrm{T}})^{-1} = (\boldsymbol{A}^{-1})^{\mathrm{T}}$. **(b)** 设 $\boldsymbol{b}$ 是 n 维向量, 那么 $\boldsymbol{Ax} = \boldsymbol{b}$ 恰有一解, 证明正规方程可以求得这个解.

7. 求经过下面各组数据点的最佳直线, 并求出 RMSE:

(a) $(-3,3),(-1,2),(0,1),(1,-1),(3,-4)$; **(b)** $(1,1),(1,2),(2,2),(2,3),(4,3)$.

8. 求经过下面各组数据点的最佳直线, 并求出 RMSE:

(a) $(0,0),(1,3),(2,3),(5,6)$; **(b)** $(1,2),(3,2),(4,1),(6,3)$;

(c) $(0,5),(1,3),(2,3),(3,1)$.

9. 求经过习题 8 中每组数据点的最佳抛物线, 并且与最佳直线拟合比较 RMSE.

10. 求经过习题 8 中每组数据点的最佳三次多项式. 另外求出这种三次插值多项式并进行比较.

11. 假设测量模型火箭的高度 4 次, 测量时间和高度是 $(t,h) = (1,135),(2,265),(3,385),(4.485)$, 单位分别是秒和米. 拟合模型 $h = a + bt - 4.905t^2$ 来估计火箭最终的最大高度以及返回地球的时间.

12. 给定数据点 $(x,y,z) = (0,0,3),(0,1,2),(1,0,3),(1,1,5),(1,2,6)$, 在三维空间中, 求最佳拟合这些数据的平面 (模型 $z = c_0 + c_1x + c_2y$).

计算机问题 4.1

1. 对下列不相容方程组构造正规方程, 并且计算最小二乘解以及 2- 范数误差:

(a) $\begin{bmatrix} 3 & -1 & 2 \\ 4 & 1 & 0 \\ -3 & 2 & 1 \\ 1 & 1 & 5 \\ -2 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ -5 \\ 15 \\ 0 \end{bmatrix}$;

(b) $\begin{bmatrix} 4 & 2 & 3 & 0 \\ -2 & 3 & -1 & 1 \\ 1 & 3 & -4 & 2 \\ 1 & 0 & 1 & -1 \\ 3 & 1 & 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 2 \\ 0 \\ 5 \end{bmatrix}$.

2. 考虑计算机问题 3.2.3 中世界石油生产数据, 求最佳最小二乘: (a) 直线; (b) 抛物线; (c) 立方曲线, 它们都通过 10 个数据点. 并求这些拟合的 RMSE. 利用上面的每一种拟合来估计 2010 年的生产水平. 就 RMSE 而言, 哪一种拟合最好地代表了这些数据?

3. 考虑计算机问题 3.1.1 中世界人口数据, 求最佳最小二乘: (a) 直线; (b) 抛物线, 它们都通过这些数据点. 并求这些拟合的 RMSE. 在每一种情形下, 估计 1980 年的人口. 哪一种拟合给出最佳估计?

4. 考虑习题 3.1.13 中二氧化碳浓度的数据, 求最佳最小二乘: (a) 直线; (b) 抛物线; (c) 立方曲线, 它们都通过这些数据点. 并求这些拟合的 RMSE. 在每种情形下, 估计 1950 年 CO_2 的浓度.

5. 一家公司在 22 个近似相等大小的城市尝试销售一种新的软饮料. 售价 (美元) 以及在城市中每周的销量如表 4-1 所示.

表 4-1

城市	售价	销量/周	城市	售价	销量/周
1	0.59	3 980	12	0.49	6 000
2	0.80	2 200	13	1.09	1 190
3	0.95	1 850	14	0.95	1 960
4	0.45	6 100	15	0.79	2 760
5	0.79	2 100	16	0.65	4 330
6	0.99	1 700	17	0.45	6 960
7	0.90	2 000	18	0.60	4 160
8	0.65	4 200	19	0.89	1 990
9	0.79	2 440	20	0.79	2 860
10	0.69	3 300	21	0.99	1 920
11	0.79	23 00	22	0.85	2 160

(a) 首先, 公司想要找出 "需求曲线": 在每一种可能价格下, 销量是多少, 设 P 表示价格, S 表示每周销量. 求在最小二乘下最佳拟合表中数据的直线 $S = c_1 + c_2P$. 求正规方程以及最小二乘直线的系数 c_1 及 c_2. 沿着数据画出最小二乘直线并且计算均方根误差.

(b) 在研究了试销的结果之后, 公司将设置一个全国统一的售价. 每件产品的制造成本给定为 0.23 美元, 总利润 (每个城市, 每周) 是 $S(P-0.23)$ 美元. 用前面最小二乘近似的结果去求使公司利润最大的售价.

6. 抛物线 $y = x^2$ 在 [0, 1] 上的 "斜率" 是什么, 求在该区间 n 个等距点上拟合这条抛物线的最小二乘直线: (a) $n = 10$; (b) $n = 20$. 画出抛物线以及这些直线. 当 $n \to \infty$ 时, 你预计结果是什么? (c) 求函数 $F(c_1, c_2) = \int_0^1 (x^2 - c_1 - c_2x)^2 \mathrm{d}x$的最小值, 并且解释它与

这个问题的关系.

7. 求经过图 3-5 中 13 个数据点的最小二乘：(a) 直线; (b) 抛物线. 并求出每种拟合的 RMSE.
8. 设 $\boldsymbol{A}$ 是由 10×10 阶 Hilbert 矩阵的前 n 列形成的 $10\times n$ 矩阵, $\boldsymbol{c}$ 是 n 维向量 $[1,\cdots,1]$, 并且令 $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{c}$. 用正规方程解最小二乘问题 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$; (a) $n=6$, (b) $n=8$, 并与准确的最小二乘解 $\bar{\boldsymbol{x}}=\boldsymbol{c}$ 进行比较. 能够计算多少位准确的小数? 用条件数来解释结果 (这个最小二乘问题还会在计算机问题 4.3.3 中出现).
9. 设 $x_1,\cdots,x_{11}$ 是在 $[2,4]$ 中的 11 个等距点以及 $y_i=1+x_i+x_i^2+\cdots+x_i^d$. 利用正规方程计算最佳 d 次多项式, 这里 (a) $d=5$, (b) $d=6$, (c) $d=8$. 与例 4.5 进行比较. 可以计算系数的多少位准确小数? 用条件数来解释这些结果 (这个最小二乘问题还会在计算机问题 4.3.4 中出现).

4.2 模型综述

前面的线性模型和多项式模型说明了最小二乘拟合数据的用处. 数据建模的艺术包括各种各样的模型, 某些来自作为数据源基础的物理原理, 而其他的则以经验因素为基础.

4.2.1 周期数据

周期数据要求周期模型. 例如外部气温在许多时间尺度下服从周期循环, 包括由地球自转以及地球绕太阳公转所确定的每天的周期以及每年的周期. 作为第一个例子, 将逐时温度 (hourly temperature) 数据与正弦及余弦拟合.

例 4.6 把在下表中列出的华盛顿地区在 2001 年 1 月 1 日的温度记录与周期模型拟合:

一天中的时间	t	温度 (°C)
午夜 12 时	0	−2.2
上午 3 时	$\frac{1}{8}$	−2.8
上午 6 时	$\frac{1}{4}$	−6.1
上午 9 时	$\frac{3}{8}$	−3.9
正午 12 时	$\frac{1}{2}$	0.0
下午 3 时	$\frac{5}{8}$	1.1
下午 6 时	$\frac{3}{4}$	−0.6
下午 9 时	$\frac{7}{8}$	−1.1

我们选择模型 $y=c_1+c_2\cos 2\pi t+c_3\sin 2\pi t$ 来匹配温度以 24 小时为周期的事实 (至少不出现较长期的温度变动下是如此). 模型利用这一信息, 把周期设定为恰好一日, 把天作为 t 的单位. 变量 t 在表中的单位为天.

把数据代入模型, 结果是以下不相容的线性方程组:

$$\begin{cases} c_1 + c_2 \cos 2\pi(0) + c_3 \sin \pi(0) = -2.2, \\ c_1 + c_2 \cos 2\pi\left(\frac{1}{8}\right) + c_3 \sin 2\pi\left(\frac{1}{8}\right) = -2.8, \\ c_1 + c_2 \cos 2\pi\left(\frac{1}{4}\right) + c_3 \sin 2\pi\left(\frac{1}{4}\right) = -6.1, \\ c_1 + c_2 \cos 2\pi\left(\frac{3}{8}\right) + c_3 \sin 2\pi\left(\frac{3}{8}\right) = -3.9, \\ c_1 + c_2 \cos 2\pi\left(\frac{1}{2}\right) + c_3 \sin 2\pi\left(\frac{1}{2}\right) = 0.0, \\ c_1 + c_2 \cos 2\pi\left(\frac{5}{8}\right) + c_3 \sin 2\pi\left(\frac{5}{8}\right) = 1.1, \\ c_1 + c_2 \cos 2\pi\left(\frac{3}{4}\right) + c_3 \sin 2\pi\left(\frac{3}{4}\right) = -0.6, \\ c_1 + c_2 \cos 2\pi\left(\frac{7}{8}\right) + c_3 \sin 2\pi\left(\frac{7}{8}\right) = -1.1. \end{cases}$$

相应的不相容矩阵方程是 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中

$$\boldsymbol{A} = \begin{bmatrix} 1 & \cos 0 & \sin 0 \\ 1 & \cos\frac{\pi}{4} & \sin\frac{\pi}{4} \\ 1 & \cos\frac{\pi}{2} & \sin\frac{\pi}{2} \\ 1 & \cos\frac{3\pi}{4} & \sin\frac{3\pi}{4} \\ 1 & \cos\pi & \sin\pi \\ 1 & \cos\frac{5\pi}{4} & \sin\frac{5\pi}{4} \\ 1 & \cos\frac{3\pi}{2} & \sin\frac{3\pi}{2} \\ 1 & \cos\frac{7\pi}{4} & \sin\frac{7\pi}{4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 1 & 0 & 1 \\ 1 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 1 & -1 & 0 \\ 1 & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ 1 & 0 & 1 \\ 1 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} -2.2 \\ -2.8 \\ -6.1 \\ -3.9 \\ 0.0 \\ 1.1 \\ -0.6 \\ -1.1 \end{bmatrix}.$$

正规方程 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ac} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ 是

$$\begin{bmatrix} 8 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} -15.6 \\ -2.977\,8 \\ -10.237\,6 \end{bmatrix},$$

容易解出 $c_1 = -1.95, c_2 = -0.744\,5$, $c_3 = -2.559\,4$. 在最小二乘意义下, 模型的最佳形式是 $y = -1.950\,0 - 0.744\,5\cos 2\pi t - 2.559\,4\sin 2\pi t$, RMSE$\approx$ 1.063. 图 4-5a 把最小二乘拟合模型与实际的逐时记录温度进行了比较.

例 4.7 把温度数据与改进的模型

$$y = c_1 + c_2 \cos 2\pi t + c_3 \sin 2\pi t + c_4 \cos 4\pi t \tag{4.9}$$

进行拟合.

亮点 正交性

通过基函数的特别选取, 可以大大地简化最小二乘问题. 例如, 在例 4.6 和例 4.7 中选取产生的正规方程组已是对角形. 正交基函数的这个性质将在第 10 章详细地进行研究. 模型 (4.9) 是一个 Fourier 展开式.

现在方程组是

$$\begin{cases} c_1 + c_2 \cos 2\pi(0) + c_3 \sin \pi(0) + c_4 \cos 4\pi(0) = -2.2, \\ c_1 + c_2 \cos 2\pi\left(\frac{1}{8}\right) + c_3 \sin 2\pi\left(\frac{1}{8}\right) + c_4 \cos 4\pi\left(\frac{1}{8}\right) = -2.8, \\ c_1 + c_2 \cos 2\pi\left(\frac{1}{4}\right) + c_3 \sin 2\pi\left(\frac{1}{4}\right) + c_4 \cos 4\pi\left(\frac{1}{4}\right) = -6.1, \\ c_1 + c_2 \cos 2\pi\left(\frac{3}{8}\right) + c_3 \sin 2\pi\left(\frac{3}{8}\right) + c_4 \cos 4\pi\left(\frac{3}{8}\right) = -3.9, \\ c_1 + c_2 \cos 2\pi\left(\frac{1}{2}\right) + c_3 \sin 2\pi\left(\frac{1}{2}\right) + c_4 \cos 4\pi\left(\frac{1}{2}\right) = 0.0, \\ c_1 + c_2 \cos 2\pi\left(\frac{5}{8}\right) + c_3 \sin 2\pi\left(\frac{5}{8}\right) + c_4 \cos 4\pi\left(\frac{5}{8}\right) = 1.1, \\ c_1 + c_2 \cos 2\pi\left(\frac{3}{4}\right) + c_3 \sin 2\pi\left(\frac{3}{4}\right) + c_4 \cos 4\pi\left(\frac{3}{4}\right) = -0.6, \\ c_1 + c_2 \cos 2\pi\left(\frac{7}{8}\right) + c_3 \sin 2\pi\left(\frac{7}{8}\right) + c_4 \cos 4\pi\left(\frac{7}{8}\right) = -1.1, \end{cases}$$

它导出以下正规方程:

$$\begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} -15.6 \\ -2.977\,8 \\ -10.237\,6 \\ 4.5 \end{bmatrix}.$$

解是 $c_1 = -1.95, c_2 = -0.744\,5, c_3 = -2.559\,4$, $c_4 = 1.125$, RMSE ≈ 0.705. 图 4-5b 表明, 扩展的模型 $y = -1.95 - 0.744\,5 \cos 2\pi t - 2.589\,4 \sin 2\pi t + 1.125 \cos 4\pi t$ 显著地改进了拟合. ◀

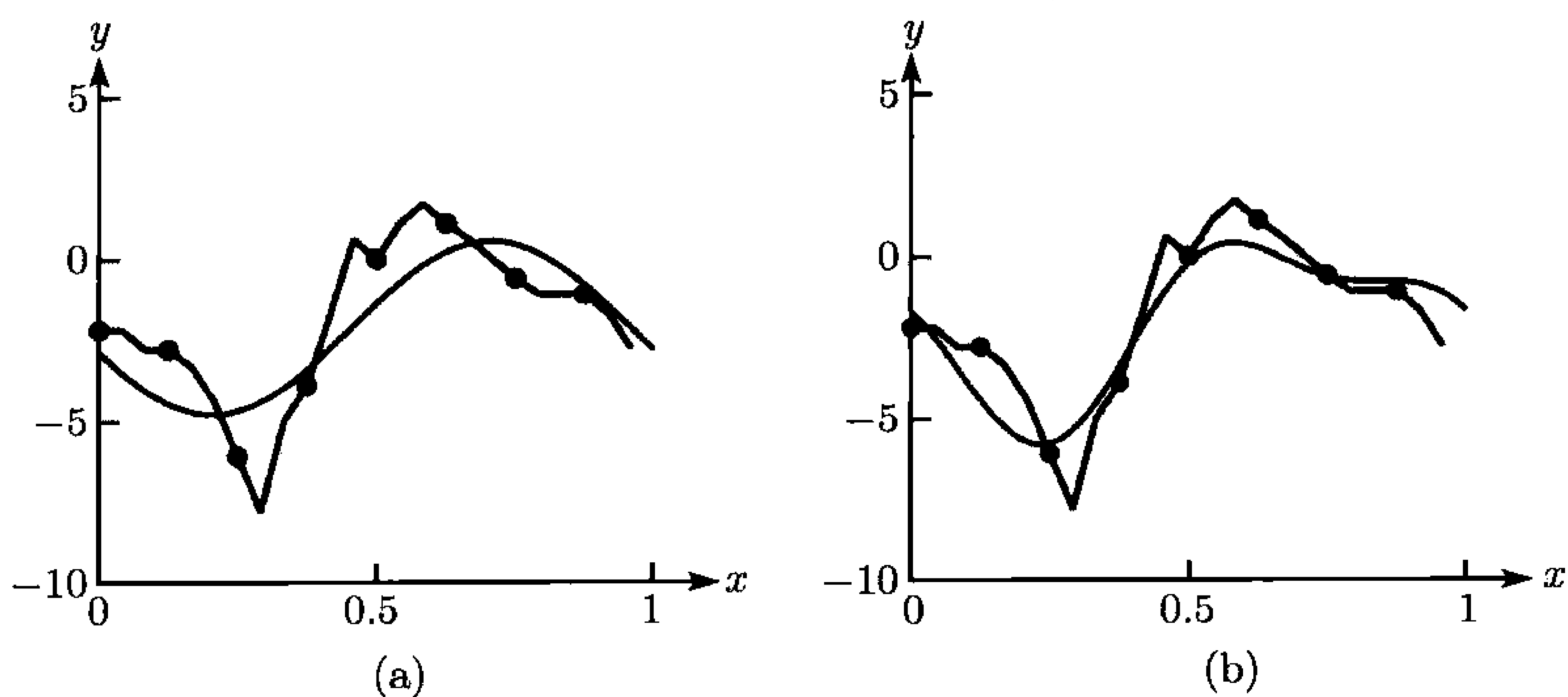


图 4-5 对例 4.6 中的周期数据的最小二乘拟合：(a) 沿着 2001 年 1 月 1 日记录的温度，正弦曲线模型 $y = -1.95 - 0.744\ 5\cos 2\pi - 2.559\ 4\sin 2\pi t$ 用粗黑体显示；(b) 改进的正弦曲线 $y = -1.95 - 0.744\ 5\cos 2\pi t - 2.559\ 4\sin 2\pi t + 1.125\cos 4\pi t$ 更接近地拟合数据

4.2.2 数据线性化

当人口变化的速度正比于它的大小时，就意味着人口的指数型增长. 在理想条件下，当增长环境没有改变以及当人口完全在环境的承载能力之下时，这种模型是一个好的代表.

指数型模型(exponential model)

$$y = c_1 e^{c_2 t}, \tag{4.10}$$

由于 c_2 不是线性地出现在模型方程中，所以它不能直接通过最小二乘进行拟合. 一旦把数据点代入模型，困难就明显了：要求解系数的方程组不是线性的，而且不能表示为线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，因此我们对正规方程的推导是不切合的.

处理非线性系数的问题有两种方法. 比较难的方法是直接极小化最小二乘误差，即解非线性最小二乘问题. 我们将在 4.4 节回到这个问题. 比较简单的方法是改变这个问题. 我们可以解一个不同的问题，它通过 “线性化” 模型而与原问题相关，而不是解原来的最小二乘问题.

在指数型模型 (4.10) 这种情形下，通过应用以下的自然算法把模型线性化:

$$\ln y = \ln(c_1 e^{c_2 t}) = \ln c_1 + c_2 t. \tag{4.11}$$

注意，对于指数型模型，$\ln y$ 的图形是线性的. 初看起来似乎我们仅仅把一个问题换成另一个. 现在系数 c_2 在模型中是线性的，但是 c_1 还不是. 然而，通过重新命名 $k = \ln c_1$，我们就可以写成

$$\ln y = k + c_2 t. \tag{4.12}$$

现在两个系数 k 及 c_2 在模型中都是线性的. 对最佳的 k 及 c_2 求解正规方程之后, 如果愿意, 我们就能够求得相应的 $c_1 = e^k$.

应该注意我们摆脱非线性系数这个困难的方法是改变问题. 我们提出的原来的最小二乘问题是对 (4.10) 拟合数据, 也就是求使得方程 $c_1e^{c_2t_i} = y_i, (i = 1, \cdots, n)$ 的残差的平方和

$$(c_1e^{c_2t_1} - y_1)^2 + \cdots + (c_1e^{c_2t_n} - y_n)^2 \tag{4.13}$$

极小化的 c_1, c_2. 而现在, 我们求解的问题是在 "对数空间" 极小化最小二乘误差, 也就是通过求使得方程 $\ln c_1 + c_2t_i = \ln y_i \ (i = 1, \cdots, n)$ 的残差平方和

$$(\ln c_1 + c_2t_1 - \ln y_1)^2 + \cdots + (\ln c_1 + c_2t_n - \ln y_n)^2 \tag{4.14}$$

极小化的 c_1, c_2. 这是两种不同的极小化, 并有不同的解, 这意味着它们一般情况下会得到系数 c_1, c_2 的不同的值.

对这个问题哪种方法正确, 是非线性最小二乘问题 (4.13) 还是模型线性化的形式 (4.14)? 前者是我们已定义的最小二乘. 后者并不是. 然而, 依据数据的前后关系, 两者中任一个都可以是更自然地选取. 要回答这个问题, 使用者需要确定哪一种误差对极小化最重要, 是原来意义下的误差还是在 "对数空间" 下的误差. 事实上, 对数模型是线性的, 因此可以说只有通过对数变换把数据变成线性关系之后评估模型的拟合才会比较自然.

例 4.8 采用模型线性化, 对表 4-2 中的世界汽车供应量数据求最佳最小二乘指数型拟合 $y = c_1e^{c_2t}$:

表 4-2

年	1950	1955	1960	1965	1970	1975	1980
辆 ($\times 10^6$)	53.05	73.04	98.31	139.78	193.48	260.20	320.39

数据描述了全世界在给定年份汽车经营的数量. 从 1950 年开始按年定义时间变量 t. 解线性最小二乘问题得到 $k_1 \approx 3.989\,6, c_2 \approx 0.061\,52$. 因为 $c_1 \approx e^{3.989\,6} \approx 54.03$, 所以模型是 $y = 53.03e^{0.061\,52t}$. 在对数空间, 对数线性化模型的 RMSE $\approx 0.035\,7$, 而原来的指数模型的 RMSE ≈ 9.56. 最佳模型及数据在图 4-6 中画出.

例 4.9 从 20 世纪 70 年代早期开始, 英特尔公司的 CPU 上的晶体管数量在表 4-3 中给出. 对这些数据进行模型 $y = c_1e^{c_2t}$ 的拟合.

通过使用模型线性化公式 (4.11), 参数将被拟合. 对模型线性化给出

$$\ln y = k + c_2t.$$

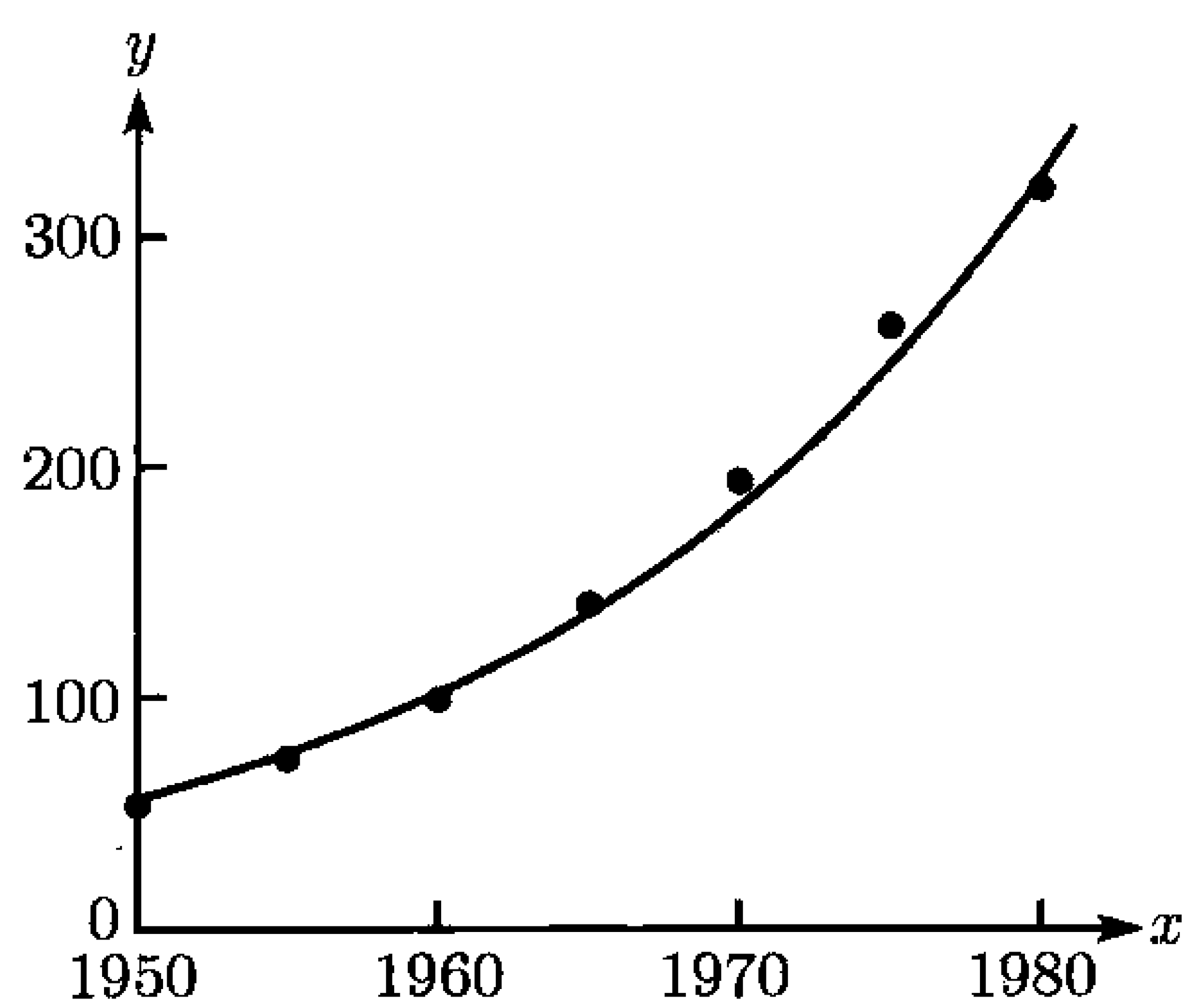


图 4-6 采用线性化, 世界汽车供应量数据的指数型拟合: 最佳最小二乘拟合是$y = 54.03e^{0.061\,52t}$

◀

表 4-3

CPU	年	晶体管数	CPU	年	晶体管数
4 004	1971	2 250	奔腾	1993	3 100 000
8 008	1972	2 500	奔腾 II	1997	7 500 000
8 080	1974	5 000	奔腾 III	1999	24 000 000
8 086	1978	29 000	奔腾 4	2000	42 000 000
286	1982	120 000	安腾	2002	220 000 000
386	1985	275 000	安腾 2	2003	410 000 000
486	1989	1 180 000			

令 $t = 0$ 对应于 1970 年. 把这些数据代入线性化模型得到

$$\begin{aligned} k + c_2(1) &= \ln 2\ 250, \\ k + c_2(2) &= \ln 2\ 500, \\ k + c_2(4) &= \ln 5\ 000, \\ k + c_2(8) &= \ln 29\ 000, \end{aligned} \tag{4.15}$$

如此等等. 矩阵方程是 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中 $\boldsymbol{x} = (k, c_2)$,

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \\ 1 & 8 \\ \vdots & \vdots \\ 1 & 33 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \ln 2\ 250 \\ \ln 2\ 500 \\ \ln 5\ 000 \\ \ln 29\ 000 \\ \vdots \\ \ln 410\ 000\ 000 \end{bmatrix}. \tag{4.16}$$

正规方程 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ax} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ 是

$$\begin{bmatrix} 13 & 235 \\ 235 & 5\ 927 \end{bmatrix} \begin{bmatrix} k \\ c_2 \end{bmatrix} = \begin{bmatrix} 176.90 \\ 3\ 793.23 \end{bmatrix},$$

它有解 $k \approx 7.197$ 以及 $c_2 \approx 0.354\,6$, 可以推出 $c_1 = \mathrm{e}^k \approx 1\,335.3$. 在图 4-7 中, 沿着数据显示了指数型曲线 $y = 1\,335.3\mathrm{e}^{0.359\,6t}$. 对这个规律加倍的时间是 $\ln \frac{2}{c_2} \approx 1.95$ 年. 英特尔的创建者之一 Gordon C. Moore 在 1965 年预言随后十年, 每两年计算效率将加倍. 令人惊奇的是, 指数型速率持续了 40 年. 在图 4-7 中存在某些证据表明, 从 2000 年开始, 这个速度已经加快.

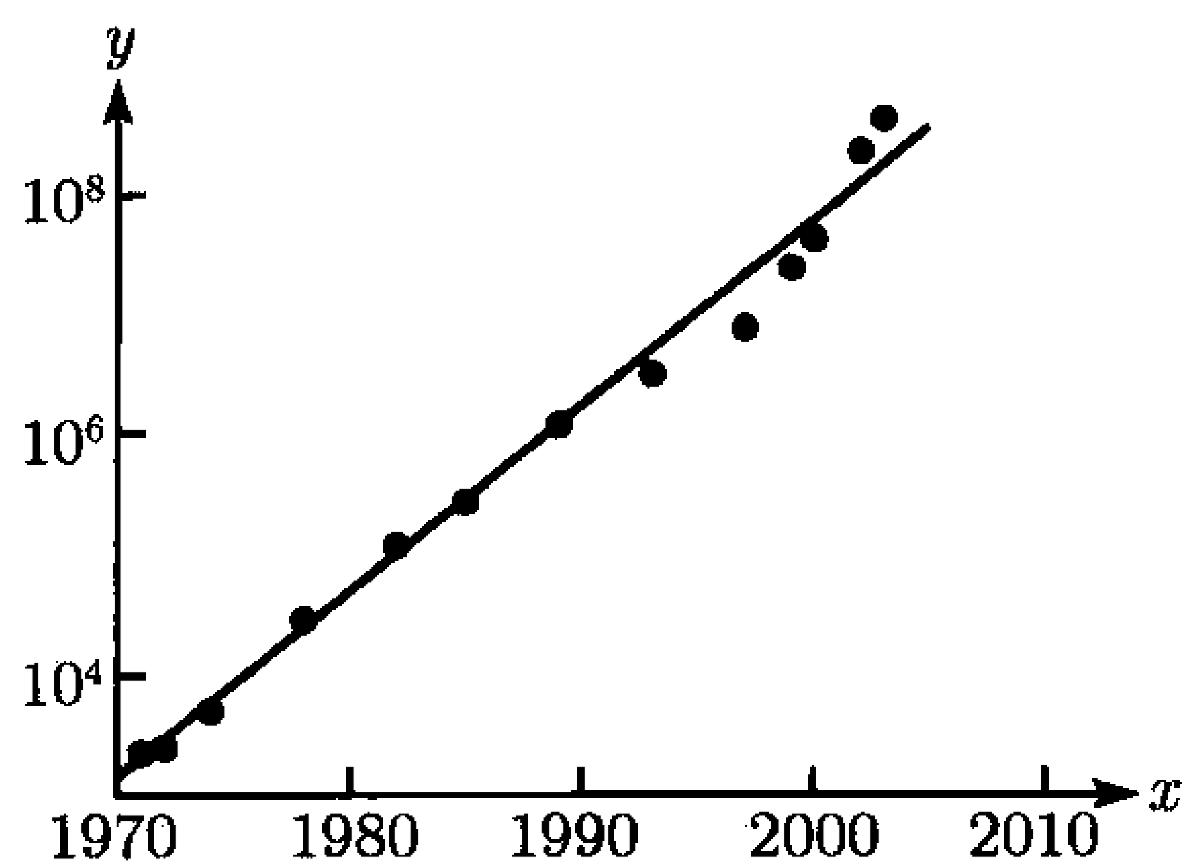


图 4-7 Moore 定律的半对数图: CPU 芯片的晶体管数量与年份的关系曲线

◀

另一个非线性系数的重要例子是幂律(power law) 模型 $y = c_1 t^{c_2}$. 这个模型也可以通过在两边取对数进行线性化而得到简化

$$\ln y = \ln c_1 + c_2 \ln t = k + c_2 \ln t. \tag{4.17}$$

把数据代入模型将给出

$$k + c_2 \ln t_1 = \ln y_1, \tag{4.18}$$

$$\vdots$$

$$k + c_2 \ln t_n = \ln y_n, \tag{4.19}$$

结果得到矩阵形式 $\boldsymbol{Ax} = \boldsymbol{b}$:

$$\boldsymbol{A} = \begin{bmatrix} 1 & \ln t_1 \\ \vdots & \vdots \\ 1 & \ln t_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \ln y_1 \\ \vdots \\ \ln y_n \end{bmatrix}. \tag{4.20}$$

正规方程得以确定 k 以及 c_2, 以及 $c_1 = \mathrm{e}^k$.

例 4.10 采用线性化把给出的身高–体重数据与幂律模型拟合.

2002 年, 由 (美国) 疾病控制中心 (CDC) 完成的《美国国家健康与营养调查》收集了 2~11 岁男孩的平均身高与体重. 结果如表 4-4 所示.

按照前面的策略, 求出体重与身高的幂律是 $W = 16.3H^{2.42}$. 在图 4-8 中画出了这种关系. 因为重量也代表了体积, 所以系数 $c_2 \approx 2.42$ 可以被看作人体的 “有效体积”.

表 4-4

年龄 (年)	身高 (m)	体重 (kg)	年龄 (年)	身高 (m)	体重 (kg)
2	0.912 0	13.7	7	1.260 0	27.2
3	0.986 0	15.9	8	1.320 0	32.7
4	1.060 0	18.5	9	1.380 0	36.0
5	1.130 0	21.3	10	1.410 0	38.6
6	1.190 0	23.5	11	1.490 0	43.7

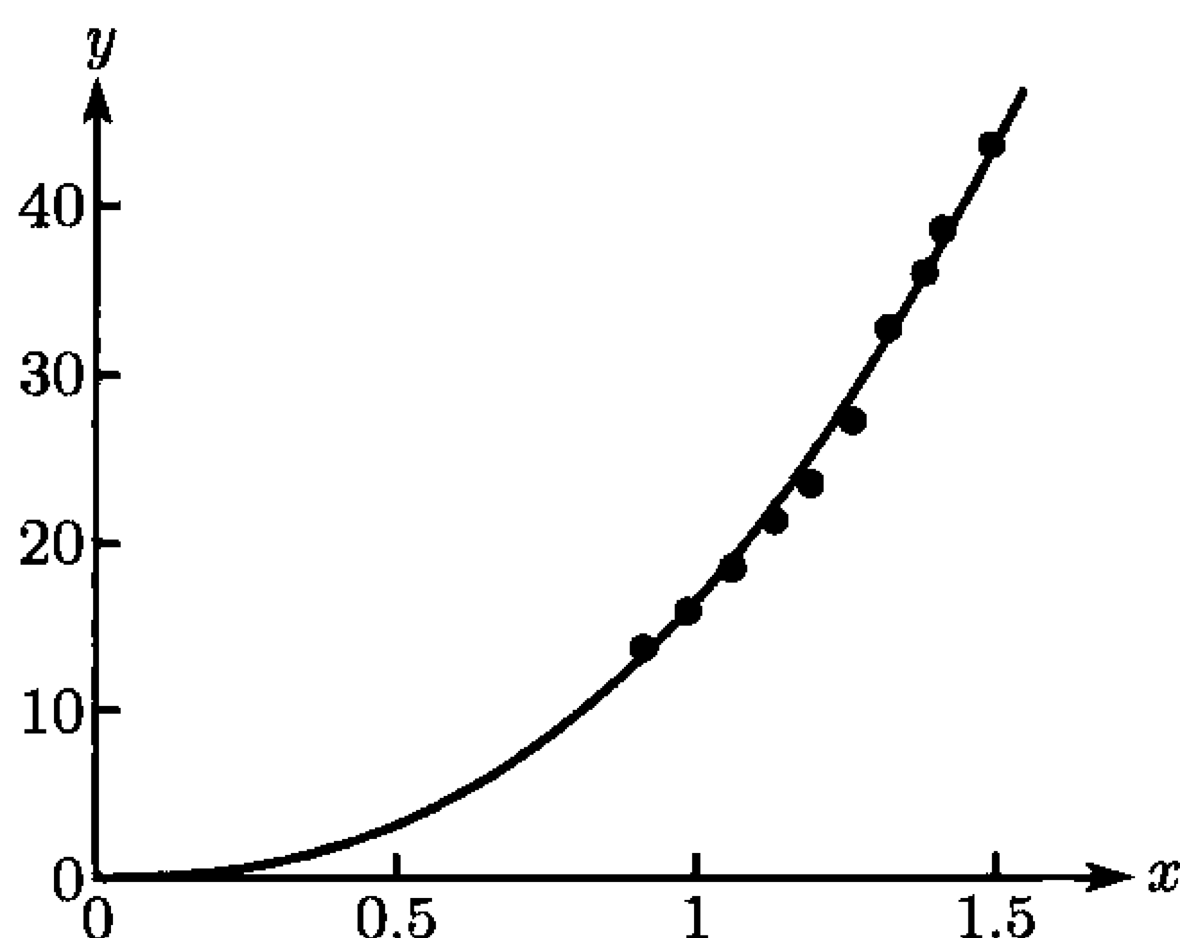


图 4-8 关于 2~11 岁的男孩体重与身高的幂律: 最佳拟合公式是 $W = 16.3H^{2.42}$

◀

在血液中, 药物浓度 y 与时间的关系由下式描述:

$$y = c_1 t e^{c_2 t}. \tag{4.21}$$

这里 t 表示药物引入机体后的时间. 这个模型的特点是当药物进入血液后迅速上升, 接着缓慢地指数型衰减. 药物的**半衰期**(half-life) 是从浓度最高点的时间到它下降一半水平的时间. 通过对二者取自然对数能把模型线性化, 得到

$$\ln y = \ln c_1 + \ln t + c_2 t,$$
$$k + c_2 t = \ln y - \ln t,$$

这里, 我们已令 $k = \ln c_1$. 这就导出矩阵方程 $\boldsymbol{Ax} = \boldsymbol{b}$, 其中

$$\boldsymbol{A} = \begin{bmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \ln y_1 - \ln t_1 \\ \vdots \\ \ln y_n - \ln t_n \end{bmatrix}. \tag{4.22}$$

解正规方程求 k 以及 c_2, 并且 $c_1 = e^k$.

例 4.11 把模型 (4.21) 与表 4-5 中给出的病人血液中药物诺氟西汀 (norfluoxetine) 的测量水平 (ng/ml) 进行拟合.

解正规方程得到 $k \approx 2.28$ 以及 $c_2 \approx -0.215$, 并且 $c_1 \approx e^{2.28} \approx 9.77$. 模型的最佳形式是 $y = 9.77te^{-0.215t}$, 画在图 4-9 中. 根据模型, 可以估计从最高浓度下降到一半水平的时间 (见计算机问题 5).

表 4-5

小时	1	2	3	4	5	6	7	8
浓度 (ng/ml)	8.0	12.3	15.5	16.8	17.1	15.8	15.2	14.0

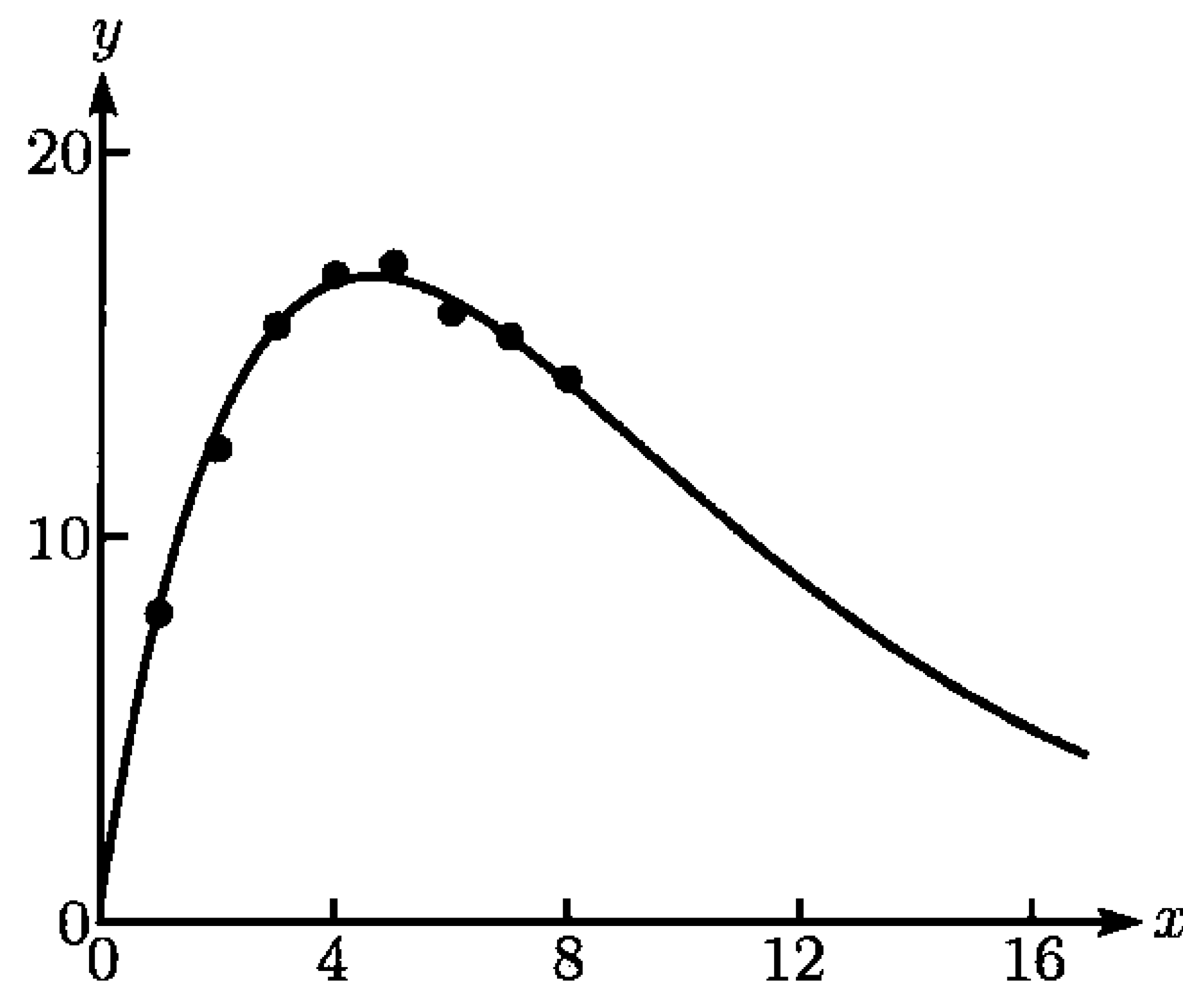


图 4-9 血液中药物浓度的曲线：模型 (4.21) 表明在初始高峰之后呈指数型衰减

◀

认识到模型线性化改变了最小二乘问题很重要. 关于线性化问题得到的解将极小化 RMSE, 这一点对原问题则不一定成立, 原问题通常会有一组不同的最优参数. 如果这些参数非线性地进入模型, 它们就不可能从正规方程解得, 因此, 我们需要非线性技术去解原来的最小二乘问题. 这在 4.4 节中的 Gauss-Newton 法中完成, 在那里我们将重新使用汽车供应量数据, 并且比较线性化与非线性形式的指数模型.

习题 4.2

1. 把数据与周期模型 $y = F_3(t) = c_1 + c_2\cos 2\pi t + c_3 \sin 2\pi t$ 进行拟合. 求出 2- 范数误差以及 RMSE.

(a)

t	y
0	1
1/4	3
1/2	2
3/4	0

(b)

t	y
0	1
1/4	3
1/2	2
3/4	1

(c)

t	y
0	3
1/2	1
1	3
3/2	2

2. 把数据与周期模型 $F_3(t) = c_1 + c_2\cos 2\pi t + c_3 \sin 2\pi t$ 以及 $F_4(t) = c_1 + c_2\cos 2\pi t + c_3 \sin 2\pi t + c_4 \cos 4\pi t$ 进行拟合, 求出 2- 范数误差 $\|e\|_2$, 并且比较 F_3 及 F_4 的拟合.

(a)

t	y
0	0
1/6	2
1/3	0
1/2	−1
2/3	1
5/6	1

(b)

t	y
0	4
1/6	2
1/3	0
1/2	−5
2/3	−1
5/6	3

3. 通过采用线性化, 把数据与指数型模型进行拟合. 求误差 $\|e\|_2$.

(a)

t	y
−2	1
0	2
1	2
2	5

(b)

t	y
0	1
1	1
1	2
2	4

4. 通过采用线性化, 把数据与指数型模型拟合. 求误差 $\|e\|_2$.

(a)

t	y
−2	4
−1	2
1	1
2	1/2

(b)

t	y
0	10
1	5
2	2
3	1

5. 通过采用线性化, 把数据与幂律模型拟合. 求拟合的 RMSE.

(a)

t	y
1	6
2	2
3	1
4	1

(b)

t	y
1	2
1	4
2	5
3	6
5	10

6. 把数据与药物浓度模型 (4.21) 拟合, 求拟合的 RMSE.

(a)

t	y
1	3
2	4
3	5
4	5

(b)

t	y
1	2
2	4
3	3
4	2

计算机问题 4.2

1. 把表 4-6 所示的 2003 年日本每个月的石油消费量数据与周期模型 (4.9) 拟合, 并且计算 RMSE.

表 4-6

月	石油消费 (10^6 桶/天)	月	石油消费 (10^6 桶/天)
1 月	6.224	7 月	4.994
2 月	6.665	8 月	5.012
3 月	6.241	9 月	5.108
4 月	5.302	10 月	5.377
5 月	5.073	11 月	5.510
6 月	5.127	12 月	6.372

2. 例 4.6 中的温度数据取自地下气象 (the Weather Underground) 网站www.wundergr-ound.com. 从你选择的地点和日期, 找出类似每小时温度的数据, 并且把它与例题的两种正弦模型进行拟合.
3. 考虑计算机问题 3.1.1 中的世界人口数据. 通过采用线性化求数据点的最佳指数模型拟合. 估计 1980 年的人口, 并求出估计误差.
4. 考虑习题 3.1.13 中的二氧化碳浓度数据. 通过采用线性化, 求 CO_2 浓度水平与其底值 (2.97×10^{-4}) 的差的最佳指数型拟合. 估计 1950 年的 CO_2 浓度, 并求出估计误差.
5. (a) 求在模型 (4.21) 中达到最大浓度的时间. (b) 在例 4.11 的模型中用一种方程解法估计半衰期.
6. 表 4-7 给出了药物注入机体后每小时测量的血液的药物浓度. 拟合模型 (4.21). 求出估计的最大值以及半衰期. 假设药物的治疗范围是 4~15ng/ml 用你选择的方程解法估计药物浓度保持在治疗水准之内的时间.

表 4-7

小时	1	2	3	4	5	6	7	8	9	10
浓度 (ng/ml)	6.2	9.5	12.3	13.9	14.6	13.5	13.3	12.7	12.4	11.9

4.3 *QR* 分解

在第 2 章, 用 ***LU*** 分解求解矩阵方程. 分解是有用的, 因为它可以把 Gauss 消去法的步骤编码. 本节将介绍 ***QR*** 分解, 把它作为一种优于正规方程的求解最小二乘问题的方法.

通过 Gram-Schmidt 正交化方法引入这种分解之后, 我们回到例 4.5. 对于这个例子, 正规方程证明是不适当的. 我们介绍 Householder 反射作为计算 $\boldsymbol{Q}$ 及 $\boldsymbol{R}$ 的更有效的方法.

4.3.1 Gram-Schmidt 正交化和最小二乘

Gram-Schmidt 方法把一组向量正交化. 给定输入一组 n 维向量, 目标是求出表示由该组数据所张成的子空间的正交坐标系. 更精确地说, 如果给出 k 个输入向量, 计算 k 个互相正交的单位向量, 它们张成与输入向量相同的子空间, 单位长度对应于欧氏长度或者 2- 范数 (4.7), 它用于整个第 4 章.

设 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k$ 是 $\mathbb{R}^n$ 中线性无关的向量. Gram-Schmidt 方法首先定义

$$\boldsymbol{y}_1 = \boldsymbol{q}_1 = \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|_2}.$$

注意

$$\boldsymbol{v}_1 = r_{11}\boldsymbol{q}_1,$$

这里 $r_{11}=\|\boldsymbol{v}_1\|_2$, $\boldsymbol{q}_1$ 是 $\boldsymbol{v}_1$ 方向的单位向量. 为了求第二个单位向量, 定义

$$\boldsymbol{y}_2=\boldsymbol{v}_2-\boldsymbol{q}_1(\boldsymbol{q}_1^{\mathrm{T}}\boldsymbol{v}_2),$$
$$\boldsymbol{q}_2=\frac{\boldsymbol{y}_2}{\|\boldsymbol{y}_2\|_2}.$$

注意

$$\boldsymbol{v}_2=\boldsymbol{y}_2+\boldsymbol{q}_1(\boldsymbol{q}_1^{\mathrm{T}}\boldsymbol{v}_2)=r_{22}\boldsymbol{q}_2+r_{12}\boldsymbol{q}_1,$$

这里 $r_{22}=\|\boldsymbol{y}_2\|_2$, $r_{12}=\boldsymbol{q}_1^{\mathrm{T}}\boldsymbol{v}_2$. 一般地, 定义

$$\boldsymbol{y}_i=\boldsymbol{v}_i-\boldsymbol{q}_1(\boldsymbol{q}_1^{\mathrm{T}}\boldsymbol{v}_i)-\boldsymbol{q}_2(\boldsymbol{q}_2^{\mathrm{T}}\boldsymbol{v}_i)-\cdots-\boldsymbol{q}_{i-1}(\boldsymbol{q}_{i-1}^{\mathrm{T}}\boldsymbol{v}_i), \tag{4.23}$$

$$\boldsymbol{q}_i=\frac{\boldsymbol{y}_i}{\|\boldsymbol{y}_i\|_2},$$

这意味着

$$\boldsymbol{v}_i=r_{ii}\boldsymbol{q}_i+r_{1i}\boldsymbol{q}_1+\cdots+r_{i-1,i}\boldsymbol{q}_{i-1},$$

这里 $r_{ii}=\|\boldsymbol{y}_i\|_2$, $r_{ji}=\boldsymbol{q}_j^{\mathrm{T}}\boldsymbol{v}_i$ $(j=1,\cdots,i)$.

显然, 每一个 $\boldsymbol{q}_i$ 正交于前面构造的正交的 $\boldsymbol{q}_j$ $(j=1,\cdots,i-1)$, 由 (4.23) 得到

$$\boldsymbol{q}_j^{\mathrm{T}}\boldsymbol{y}_i=\boldsymbol{q}_j^{\mathrm{T}}\boldsymbol{v}_i-\boldsymbol{q}_j^{\mathrm{T}}\boldsymbol{q}_1\boldsymbol{q}_1^{\mathrm{T}}\boldsymbol{v}_i-\cdots-\boldsymbol{q}_j^{\mathrm{T}}\boldsymbol{q}_{i-1}\boldsymbol{q}_{i-1}^{\mathrm{T}}\boldsymbol{v}_i=0.$$

从几何上来讲, (4.23) 对应于从 $\boldsymbol{v}_i$ 中减去 $\boldsymbol{v}_i$ 在前面确定的正交向量 $\boldsymbol{q}_j$ $(j=1,\cdots,i-1)$ 上的投影. 所余下来的显然正交于 $\boldsymbol{q}_j$, 并且, 除以本身的长度后变成单位向量, 被用作 $\boldsymbol{q}_i$.

例 4.12 对 $\boldsymbol{A}=\begin{bmatrix}1 & -4\\ 2 & 3\\ 2 & 2\end{bmatrix}$ 的列应用 Gram-Schmidt 正交化.

设 $\boldsymbol{y}_1=\boldsymbol{v}_1=\begin{bmatrix}1\\2\\2\end{bmatrix}$. 于是 $r_{11}=\|\boldsymbol{y}_1\|_2=\sqrt{1^2+2^2+2^2}=3$, 并且第一个单位向量是

$$\boldsymbol{q}_1=\frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|_2}=\begin{bmatrix}\frac{1}{3}\\ \frac{2}{3}\\ \frac{2}{3}\end{bmatrix}.$$

为了求第二个单位向量, 设

$$\boldsymbol{y}_2 = \boldsymbol{v}_2 - \boldsymbol{q}_1\boldsymbol{q}_1^{\mathrm{T}}\boldsymbol{v}_2 = \begin{bmatrix} -4 \\ 3 \\ 2 \end{bmatrix} - \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix} \times 2 = \begin{bmatrix} -\frac{14}{3} \\ \frac{5}{3} \\ \frac{2}{3} \end{bmatrix},$$

$$\boldsymbol{q}_2 = \frac{\boldsymbol{y}_2}{\|\boldsymbol{y}_2\|_2} = \frac{1}{5}\begin{bmatrix} -\frac{14}{3} \\ \frac{5}{3} \\ \frac{2}{3} \end{bmatrix} = \begin{bmatrix} -\frac{14}{15} \\ \frac{1}{3} \\ \frac{2}{15} \end{bmatrix}.$$

按前面的记号 $r_{12} = 2, r_{22} = 5$. ◀

Gram-Schmidt 正交化的结果可以写成下列矩阵形式:

$$(\boldsymbol{v}_1|\cdots|\boldsymbol{v}_k) = (\boldsymbol{q}_1|\cdots|\boldsymbol{q}_k)\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1k} \\ & r_{22} & \cdots & r_{2k} \\ & & \ddots & \vdots \\ & & & r_{kk} \end{bmatrix}. \tag{4.24}$$

向量 $\boldsymbol{v}_i$ 线性无关的假设保证系数 r_{ii} 非零. 相反, 如果 $\boldsymbol{v}_i$ 落在 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{i-1}$ 所张成的空间, 那么在后面这些向量上的投影构成完整的向量, 并且 $r_{ii} = \|\boldsymbol{y}_i\| = 0$, 在这种情形下, Gram-Schmidt 方法终止.

当这种方法成功时, 习惯地把正交单位向量矩阵扩充为 $\mathbb{R}^n$ 的一组完备基. 例如, 可以这样做, 通过把 $n-k$ 个额外的向量加到 $\boldsymbol{v}_i$ 使得这 n 个向量张成 $\mathbb{R}^n$, 并且执行 Gram-Schmidt 方法. 使用由 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_n$ 形成的 $\mathbb{R}^n$ 的这组基, 原来的向量可以表示为

$$(\boldsymbol{v}_1|\cdots|\boldsymbol{v}_k) = (\boldsymbol{q}_1|\cdots|\boldsymbol{q}_n)\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1k} \\ & r_{22} & \cdots & r_{2k} \\ & & \ddots & \vdots \\ & & & r_{kk} \\ 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}. \tag{4.25}$$

这个矩阵方程就是由原来输入向量形成的矩阵 $\boldsymbol{A} = (\boldsymbol{v}_1|\cdots|\boldsymbol{v}_k)$ 的 $\boldsymbol{QR}$ **分解**. 注意矩阵的规格: $\boldsymbol{A}$ 是 $n \times k, \boldsymbol{Q}$ 是 $n \times n$, 上三角矩阵 $\boldsymbol{R}$ 是 $n \times k$, 规格与 $\boldsymbol{A}$ 相同. $\boldsymbol{QR}$ 分解中的矩阵 $\boldsymbol{Q}$ 在数值分析中具有特殊地位, 因此被赋予特殊的定义.

定义 4.1 如果 $Q^{\mathrm{T}} = Q^{-1}$, 那么方阵 Q 是正交的.

注意, 方阵正交的充要条件是它的列向量是两两正交的单位向量 (习题 7). 因此, QR 分解就是等式 $A = QR$, 这里 Q 是正交方阵, 而 R 是与 A 同阶的上三角矩阵.

正交矩阵的关键性质是保持了向量的欧氏范数.

引理 4.2 如果 Q 是 $n \times n$ 正交矩阵, x 是 n 维向量, 那么 $\|Qx\|_2 = \|x\|_2$.

证 $\|Qx\|_2 = (Qx)^{\mathrm{T}}Qx = x^{\mathrm{T}}Q^{\mathrm{T}}Qx = x^{\mathrm{T}}x = \|x\|_2$.

两个正交 $n \times n$ 矩阵的乘积仍然是正交的 (习题 8). 使用 Gram-Schmidt 方法, 一个 $n \times n$ 矩阵的 QR 分解大概需要 n^3 次乘/除法 (是 LU 分解的 3 倍), 再加上大约相同次数的加法 (习题 9).

亮点 正交性

在第 2 章我们发现 LU 分解是对 Gauss 消去法的信息进行编码的有效方法. 以相同的方法, QR 分解记录了矩阵的正交化, 即构造了张成 A 的列向量空间的一组正交向量. 用正交矩阵进行计算是最好的. 这是因为 (1) 根据定义容易求它们的逆, (2) 根据引理 4.2, 它们不放大误差.

例 4.13 求 $A = \begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix}$ 的 QR 分解.

在例 4.12 中, 我们求解正交单位向量 $q_1 = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}$, $q_2 = \begin{bmatrix} -\frac{14}{15} \\ \frac{1}{3} \\ \frac{2}{15} \end{bmatrix}$. 加上第 3 个向量 $v_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ 导出

$$y_3 = v_3 - q_1 q_1^{\mathrm{T}} v_3 - q_2 q_2^{\mathrm{T}} v_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix} \frac{1}{3} - \begin{bmatrix} -\frac{14}{15} \\ \frac{1}{3} \\ \frac{2}{15} \end{bmatrix} \left(-\frac{14}{15}\right)$$

$$= \frac{2}{225} \begin{bmatrix} 2 \\ 10 \\ -11 \end{bmatrix}, \quad q_3 = y_3/\|y_3\| = \begin{bmatrix} \frac{2}{15} \\ \frac{10}{15} \\ -\frac{11}{15} \end{bmatrix}.$$

把各部分放到一起, 我们就得到 QR 分解

$$\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} = QR = \frac{1}{15}\begin{bmatrix} 5 & -14 & 2 \\ 10 & 5 & 10 \\ 10 & 2 & -11 \end{bmatrix}\begin{bmatrix} 3 & 2 \\ 0 & 5 \\ 0 & 0 \end{bmatrix}.$$

◀

MATLAB命令qr 对 $m \times n$ 矩阵执行 QR 分解. 它不采用 Gram-Schmidt 正交化, 但是 4.3.2 节将介绍更有效和稳定的方法. 命令

```
>>[Q,R]=qr(A)
```

返回因子.

QR 分解有 3 种主要应用. 我们在这里将陈述其中两种, 第 3 种是在第 12 章介绍的关于特征值计算的 QR 算法.

首先, 可以用 QR 分解去解含 n 个未知量、n 个方程的方程组 $Ax = b$. 只需分解 $A = QR$, 因此方程 $Ax = b$ 变成 $QRx = b$ 以及 $Rx = Q^{\mathrm{T}}b$. 假设 A 非奇异, 上三角矩阵 R 的对角线元素非零, 所以 R 非奇异. 三角逆代换便得到解 x. 如前所述, 这种方法就复杂性而言花费的时间大约是 LU 分解方法的 3 倍.

第二种应用是对于最小二乘. 设 A 是 $m \times n$ 矩阵 $(m \geqslant n)$. 要把 $\|Ax - b\|_2$ 极小化, 重新写成 $\|QRx - b\|_2 = \|Rx - Q^{\mathrm{T}}b\|_2$ (根据引理 4.2). 在 Euclid 范数里面的向量是

$$\begin{bmatrix} e_1 \\ \vdots \\ e_n \\ \hdashline e_{n+1} \\ \vdots \\ e_m \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \\ \hdashline 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} d_1 \\ \vdots \\ d_n \\ \hdashline d_{n+1} \\ \vdots \\ d_m \end{bmatrix}, \tag{4.26}$$

其中 $d = Q^{\mathrm{T}}b$. 假设 $r_{ii} \neq 0$. 于是误差向量 e 的上面部分 $(e_1, \cdots, e_n)$ 通过回代能变为零, x_i 的选取对误差向量的下面部分没有影响; 显然, $(e_{n+1}, \cdots, e_m) = (d_{n+1}, \cdots, d_m)$. 因此, 通过用回解上面部分而得出的 x, 使得最小二乘解极小化, 而且最小二乘平方误差是

$$\|e\|_2 = d_{n+1}^2 + \cdots + d_m^2.$$

用 QR 分解的最小二乘

给定 $m \times n$ 不相容方程组

$$Ax = b,$$

分解 $A = QR$, 并设

$$\hat{R} = R \text{ 的上 } n \times n \text{ 子矩阵},$$

$$\hat{d} = d = Q^{\mathrm{T}} b \text{ 的前 } n \text{ 分量},$$

解 $\hat{R}\bar{x} = \hat{d}$ 得到最小二乘解 $\bar{x}$.

例 4.14 用 QR 分解解最小二乘问题 $\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix}$.

我们需要解 $Rx = Q^{\mathrm{T}} b$, 或者

$$\begin{bmatrix} 3 & 2 \\ 0 & 5 \\ \hdashline 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} 5 & 10 & 10 \\ -14 & 5 & 2 \\ 2 & 10 & -11 \end{bmatrix} \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix} = \begin{bmatrix} 15 \\ 9 \\ \hdashline 3 \end{bmatrix}.$$

亮点 条件作用

在第 2 章, 我们发现处理病态问题的最好方法是避免它们. 例 4.15 是此忠告的一个典型情形. 而例 4.5 的正规方程是病态的. QR 方法解最小二乘问题并没有构造 $A^{\mathrm{T}}A$.

最小二乘误差将是 $\|e\|_2 = \|(0,0,3)\|_2 = 3$. 使上面部分相等得到

$$\begin{bmatrix} 3 & 2 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 15 \\ 9 \end{bmatrix},$$

其解为 $\bar{x}_1 = 3.8, \bar{x}_2 = 1.8$. 这个最小二乘问题通过例 4.2 中的正规方程求得其解.

最后回到例 4.5 中的问题, 它导出病态正规方程组.

例 4.15 用 QR 分解求解例 4.5 的最小二乘问题.

在解这个含 8 个变量和 11 个方程的最小二乘问题的过程中, 正规方程格外不成功. 我们用 MATLAB 的qr命令执行另一种方法:

```
>> x=(2+(0:10)/5)';
>> y=1+x+x.^2+x.^3+x.^4+x.^5+x.^6+x.^7;
>> A=[x.^0 x x.^2 x.^3 x.^4 x.^5 x.^6 x.^7];
>> [Q,R]=qr(A);
>> b=Q'*y;
>> c=R(1:8,1:8)\b(1:8)
```

```
c=
   0.99999991014308
   1.00000021004107
   0.99999979186557
   1.00000011342980
   0.99999996325039
   1.00000000708455
   0.99999999924685
   1.00000000003409
```

用 $\boldsymbol{QR}$ 分解求得 6 位小数的准确解 $\boldsymbol{c} = [1, \cdots, 1]$. 这种方法发现最小二乘解不需形成条件数约为 10^{19} 的正规方程.

4.3.2 Householder 反射

尽管 Gram-Schmidt 正交化方法是计算矩阵的 $\boldsymbol{QR}$ 分解的一种方法, 但是它不是最好的方法. 另一种用 Householder 反射的方法需要较少的运算而且更稳定 (在舍入误差放大的意义下). 本节将定义这种反射并且说明如何利用它们分解矩阵.

Householder 反射是一个正交矩阵, 它通过一个 $m-1$ 维平面反射所有的 m 维向量. 这意味着每一个向量乘以这个矩阵时, 它的长度没有改变, 因此 Householder 反射非常适合移动向量. 给定向量 $\boldsymbol{x}$, 我们想重新确定一个相同长度的向量 $\boldsymbol{w}$, Householder 反射的诀窍是给出一矩阵 $\boldsymbol{H}$, 使得 $\boldsymbol{Hx} = \boldsymbol{w}$.

在图 4-10 中, 这种方法的由来是清楚的. 画一个 $m-1$ 维平面平分 $\boldsymbol{x}$ 和 $\boldsymbol{w}$, 并且垂直于连结它们的向量, 然后通过这个平面反射.

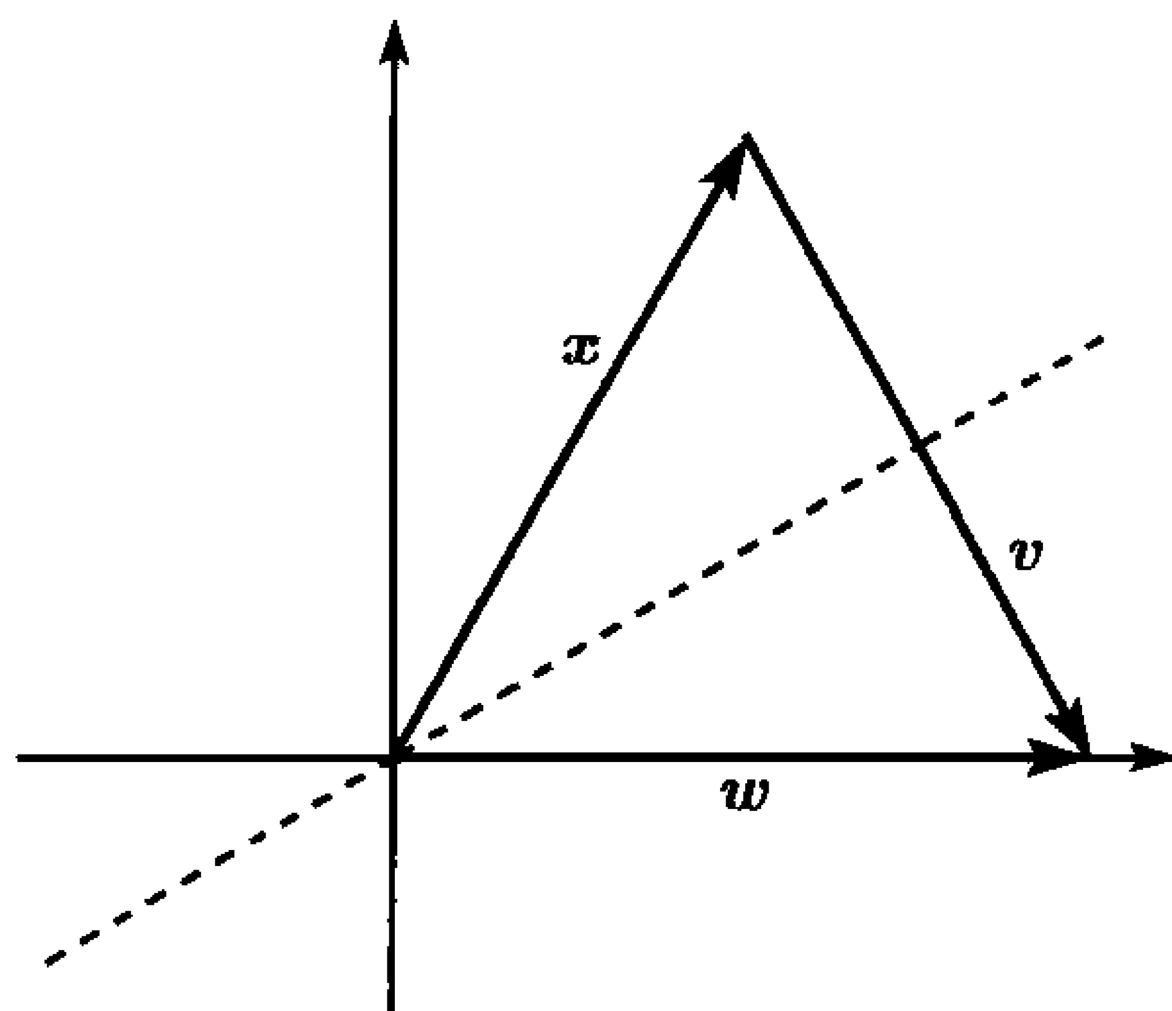


图 4-10 Householder 反射: 给出长度相同的向量 $\boldsymbol{x}$ 和 $\boldsymbol{w}$, 通过它们夹角的角平分线 (虚线) 进行的反射交换它们

引理 4.3 假设 $\boldsymbol{x}$ 和 $\boldsymbol{\omega}$ 是 Euclid 长度相同的向量, $\|\boldsymbol{x}\|_2 = \|\boldsymbol{\omega}\|_2$. 那么 $\boldsymbol{\omega} - \boldsymbol{x}$ 与 $\boldsymbol{\omega} + \boldsymbol{x}$ 正交.

证 $(\boldsymbol{\omega} - \boldsymbol{x})^{\mathrm{T}}(\boldsymbol{\omega} + \boldsymbol{x}) = \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\omega} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\omega} + \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} = \|\boldsymbol{\omega}\|^2 - \|\boldsymbol{x}\|^2 = 0.$

定义向量 $\boldsymbol{v}=\boldsymbol{\omega}-\boldsymbol{x}$, 并且考虑投影矩阵

$$\boldsymbol{P}=\frac{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}}{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}. \tag{4.27}$$

投影矩阵(projection matrix) 是满足 $\boldsymbol{P}^2=\boldsymbol{P}$ 的矩阵. 习题 11 要求读者证明 (4.27) 中的 $\boldsymbol{P}$ 是对称投影矩阵, 而且 $\boldsymbol{P}\boldsymbol{v}=\boldsymbol{v}$. 在几何上, 对任一向量 $\boldsymbol{u}$, $\boldsymbol{P}\boldsymbol{u}$ 是 $\boldsymbol{u}$ 在 $\boldsymbol{v}$ 上的投影. 图 4-10 暗示, 如果从 $\boldsymbol{x}$ 两次减去投影 $\boldsymbol{P}\boldsymbol{x}$, 我们将得到 $\boldsymbol{\omega}$. 为了证明这一点, 设 $\boldsymbol{H}=\boldsymbol{I}-2\boldsymbol{P}$, 于是

$$\begin{aligned}\boldsymbol{H}\boldsymbol{x}&=\boldsymbol{x}-2\boldsymbol{P}\boldsymbol{x}=\boldsymbol{\omega}-\boldsymbol{v}-\frac{2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{x}}{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}=\boldsymbol{\omega}-\boldsymbol{v}-\frac{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{x}}{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}-\frac{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}(\boldsymbol{\omega}-\boldsymbol{v})}{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}\\&=\boldsymbol{\omega}-\frac{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}(\boldsymbol{\omega}+\boldsymbol{x})}{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}=\boldsymbol{\omega},\end{aligned} \tag{4.28}$$

最后的等式来自引理 4.3, 这里因为 $\boldsymbol{\omega}+\boldsymbol{x}$ 正交于 $\boldsymbol{v}=\boldsymbol{\omega}-\boldsymbol{x}$.

矩阵 $\boldsymbol{H}$ 称为**Householder 反射**. 注意 $\boldsymbol{H}$ 是对称 (习题 12) 而且正交的矩阵, 因为

$$\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}=\boldsymbol{H}\boldsymbol{H}=(\boldsymbol{I}-2\boldsymbol{P})^{\mathrm{T}}(\boldsymbol{I}-2\boldsymbol{P})=\boldsymbol{I}-4\boldsymbol{P}+4\boldsymbol{P}^2=\boldsymbol{I}.$$

这些事实总结在下述定理中:

定理 4.4　Householder 反射. *设向量 $\boldsymbol{x}$ 和 $\boldsymbol{\omega}$ 满足 $\|\boldsymbol{x}\|_2=\|\boldsymbol{\omega}\|_2$. 于是存在向量 $\boldsymbol{v}$ 使得 $\boldsymbol{H}=\boldsymbol{I}-2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}/\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}$ 是对称正交矩阵, 而且 $\boldsymbol{H}\boldsymbol{x}=\boldsymbol{\omega}$. 事实上, $\boldsymbol{v}=\boldsymbol{\omega}-\boldsymbol{x}$.*

例 4.16　设 $\boldsymbol{x}=(3,4),\boldsymbol{\omega}=(5,0)$. 求满足 $\boldsymbol{H}\boldsymbol{x}=\boldsymbol{\omega}$ 的 Householder 反射 $\boldsymbol{H}$.

设

$$\boldsymbol{v}=\boldsymbol{\omega}-\boldsymbol{x}=\begin{bmatrix}5\\0\end{bmatrix}-\begin{bmatrix}3\\4\end{bmatrix}=\begin{bmatrix}2\\-4\end{bmatrix},$$

并定义投影矩阵

$$\boldsymbol{P}=\frac{\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}}{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}=\frac{1}{20}\begin{bmatrix}4&-8\\-8&16\end{bmatrix}=\begin{bmatrix}0.2&-0.4\\-0.4&0.8\end{bmatrix},$$

于是

$$\boldsymbol{H}=\boldsymbol{I}-2\boldsymbol{P}=\begin{bmatrix}1&0\\0&1\end{bmatrix}-\begin{bmatrix}0.4&-0.8\\-0.8&1.6\end{bmatrix}=\begin{bmatrix}0.6&0.8\\0.8&-0.6\end{bmatrix}.$$

检验 $\boldsymbol{H}$ 把 $\boldsymbol{x}$ 变成 $\boldsymbol{\omega}$, 而且反之亦真:

$$\boldsymbol{H}\boldsymbol{x}=\begin{bmatrix}0.6&0.8\\0.8&-0.6\end{bmatrix}\begin{bmatrix}3\\4\end{bmatrix}=\begin{bmatrix}5\\0\end{bmatrix}=\boldsymbol{\omega},$$

$$H\omega=\begin{bmatrix}0.6 & 0.8\\0.8 & -0.6\end{bmatrix}\begin{bmatrix}5\\0\end{bmatrix}=\begin{bmatrix}3\\4\end{bmatrix}=\boldsymbol{x}.$$

◀

作为 Householder 反射的第一种应用, 我们将建立一种新的方法进行 QR 分解. 在第 12 章, 我们把 Householder 应用到特征值问题, 赋于矩阵上 Hessenberg 形式. 在这两种应用中, 我们将为了唯一的目的应用反射: 把列向量 x 移到坐标轴, 作为把零放入矩阵的一种方法.

从我们想要写成形式 $A = QR$ 的矩阵 A 开始, 设 x_1 是 A 的第一列. 令 $\omega = \pm(\|x_1\|_2, 0, \cdots, 0)$ 是沿着第一坐标轴, 具有相同 Euclid 长度的向量. (在理论上, 任一符号都行. 为了最佳结果, 符号选为与 x 的第一分量的符号相反的符号, 以免在形成 v 时两个几乎相等的数相减的可能性.) 构造 Householder 反射 H_1, 使得 $H_1x = \omega$. H_1 乘以 A 得到

$$H_1A=H_1\begin{bmatrix}\times&\times&\times\\\times&\times&\times\\\times&\times&\times\\\times&\times&\times\end{bmatrix}=\begin{bmatrix}\times&\times&\times\\0&\times&\times\\0&\times&\times\\0&\times&\times\end{bmatrix}.$$

我们已把一些零引入 A. 我们想以这种方式继续下去直到 A 变成上三角矩阵, 那时我们将有了 QR 分解中的 R. 再求 Householder 反射 $\hat{H}_2$, 它把 H_1A 第二列中的下面 $m-1$ 个元素组成的 $m-1$ 维向量 x_2 变成 $\pm(\|x_2\|_2, 0, \cdots, 0)$. 由于 $\hat{H}_2$ 是 $(m-1)\times(m-1)$ 矩阵, 故定义 H_2 是 $m\times m$ 矩阵, 它是通过把 $\hat{H}_2$ 放到一个 $m\times m$ 单位矩阵的右下方而形成的. 于是

$$\left[\begin{array}{c|ccc}1&0&0&0\\\hline 0&&&\\0&&\hat{H}_2&\\0&&&\end{array}\right]=\left[\begin{array}{c|cc}\times&\times&\times\\\hline 0&\times&\times\\0&\times&\times\\0&\times&\times\end{array}\right]=\left[\begin{array}{c|cc}\times&\times&\times\\\hline 0&\times&\times\\0&0&\times\\0&0&\times\end{array}\right].$$

H_2H_1A 的结果是上三角形的第一步, 再进行一步就给出

$$\left[\begin{array}{cc|cc}1&0&0&0\\0&1&0&0\\\hline 0&0&&\\0&0&\multicolumn{2}{c}{\hat{H}_3}\end{array}\right]=\left[\begin{array}{cc|c}\times&\times&\times\\0&\times&\times\\\hline 0&0&\times\\0&0&\times\end{array}\right]=\left[\begin{array}{cc|c}\times&\times&\times\\0&\times&\times\\\hline 0&0&\times\\0&0&0\end{array}\right],$$

因此结果是

$$H_3H_2H_1A=R,$$

它是一个上三角矩阵. 左乘 n 个 Householder 反射的逆, 可以把这个结果重新写成

$$\boldsymbol{A} = \boldsymbol{H}_1\boldsymbol{H}_2\boldsymbol{H}_3\boldsymbol{R} = \boldsymbol{Q}\boldsymbol{R},$$

其中 $\boldsymbol{Q} = \boldsymbol{H}_1\boldsymbol{H}_2\boldsymbol{H}_3$. 注意 $\boldsymbol{H}_i^{-1} = \boldsymbol{H}_i$, 这是因为 $\boldsymbol{H}_i$ 是对称正交的. 计算机问题 1 要求读者写出代码表示借助于 Householder 反射的分解.

例 4.17 用 Householder 反射求 $\boldsymbol{A} = \begin{bmatrix} 3 & 1 \\ 4 & 3 \end{bmatrix}$ 的 $\boldsymbol{QR}$ 分解.

需要求出把第一列 (3, 4) 变到 x 轴上的 Householder 反射. 我们在例 4.16 中找到了这种反射 $\boldsymbol{H}_1$, 并且

$$\boldsymbol{H}_1\boldsymbol{A} = \begin{bmatrix} 0.6 & 0.8 \\ 0.8 & -0.6 \end{bmatrix}\begin{bmatrix} 3 & 1 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 0 & -1 \end{bmatrix}.$$

两边左乘 $\boldsymbol{H}_1^{-1} = \boldsymbol{H}_1$ 得到

$$\boldsymbol{A} = \begin{bmatrix} 3 & 1 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 0.6 & 0.8 \\ 0.8 & -0.6 \end{bmatrix}\begin{bmatrix} 5 & 3 \\ 0 & -1 \end{bmatrix} = \boldsymbol{QR},$$

这里 $\boldsymbol{Q} = \boldsymbol{H}_1^{\mathrm{T}} = \boldsymbol{H}_1$. ◀

例 4.18 用 Householder 反射求 $\boldsymbol{A} = \begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix}$ 的 $\boldsymbol{QR}$ 分解.

需要求出把第一列 $\boldsymbol{x} = [1, 2, 2]$ 变到向量 $\boldsymbol{\omega} = [\|\boldsymbol{x}\|_2, 0, 0]$ 的 Householder 反射. 设 $\boldsymbol{v} = \boldsymbol{\omega} - \boldsymbol{x} = [3, 0, 0] - [1, 2, 2] = [2, -2, -2]$. 参考定理 4.4, 我们有

$$\boldsymbol{H}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{2}{12}\begin{bmatrix} 4 & -4 & -4 \\ -4 & 4 & 4 \\ -4 & 4 & 4 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & -\frac{2}{3} \\ \frac{2}{3} & -\frac{2}{3} & \frac{1}{3} \end{bmatrix},$$

$$\boldsymbol{H}_1\boldsymbol{A} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & -\frac{2}{3} \\ \frac{2}{3} & -\frac{2}{3} & \frac{1}{3} \end{bmatrix}\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 0 & -3 \\ 0 & -4 \end{bmatrix}.$$

余下的步骤是把向量 $\hat{\boldsymbol{x}} = [-3, -4]$ 变成 $\hat{\boldsymbol{\omega}} = [5, 0]$. 从定理 4.4 计算 $\hat{\boldsymbol{H}}_2$ 便得到

$$\begin{bmatrix} -0.6 & -0.8 \\ -0.8 & 0.6 \end{bmatrix}\begin{bmatrix} -3 \\ -4 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix},$$

这就导出

$$H_2H_1A=\begin{bmatrix}1&0&0\\0&-0.6&-0.8\\0&-0.8&0.6\end{bmatrix}\begin{bmatrix}\frac{1}{3}&\frac{2}{3}&\frac{2}{3}\\\frac{2}{3}&\frac{1}{3}&-\frac{2}{3}\\\frac{2}{3}&-\frac{2}{3}&\frac{1}{3}\end{bmatrix}\begin{bmatrix}1&-4\\2&3\\2&2\end{bmatrix}=\begin{bmatrix}3&2\\0&5\\0&0\end{bmatrix}=R.$$

两边左乘 $H_1^{-1}H_2^{-1}=H_1H_2$ 得到 QR 分解:

$$\begin{bmatrix}1&-4\\2&3\\2&2\end{bmatrix}=H_1H_2R=\begin{bmatrix}\frac{1}{3}&\frac{2}{3}&\frac{2}{3}\\\frac{2}{3}&\frac{1}{3}&-\frac{2}{3}\\\frac{2}{3}&-\frac{2}{3}&\frac{1}{3}\end{bmatrix}\begin{bmatrix}1&0&0\\0&-0.6&-0.8\\0&-0.8&0.6\end{bmatrix}\begin{bmatrix}3&2\\0&5\\0&0\end{bmatrix}$$

$$=\frac{1}{15}\begin{bmatrix}5&-14&-2\\10&5&-10\\10&2&11\end{bmatrix}\begin{bmatrix}3&2\\0&5\\0&0\end{bmatrix},$$

于是

$$Q=\frac{1}{15}\begin{bmatrix}5&-14&-2\\10&5&-10\\10&2&11\end{bmatrix},\qquad R=\begin{bmatrix}3&2\\0&5\\0&0\end{bmatrix}.$$

检验 $A=QR$, 并与来自例 4.13 中 Gram-Schmidt 正交化的分解进行比较. ◀

对于给定的 $m\times n$ 矩阵 A, QR 分解不是唯一的. 例如, 定义 $D=\mathrm{diag}(d_1,\cdots,d_m)$, 其中每一个 d_i 或者是 $+1$ 或者是 -1. 于是 $A=QR=QDDR$, 而且我们检验 QD 是正交的, DR 是上三角的.

习题 10 要求用 Householder 反射的 QR 分解的运算计数, 它达到 $(2/3)n^3$ 次乘法以及相同次数的加法——没有像 Gram-Schmidt 那样复杂. 更重要的是, Householder 方法被认为提供了较好的单位向量的正交性而且需要的内存较少. 因此, 这是分解典型矩阵为 QR 所选择的方法.

习题 4.3

1. 用 Gram-Schmidt 正交化, 求下列矩阵的 QR 分解:

(a) $\begin{bmatrix}4&0\\3&1\end{bmatrix}$; **(b)** $\begin{bmatrix}1&2\\1&1\end{bmatrix}$; **(c)** $\begin{bmatrix}2&1\\1&-1\\2&1\end{bmatrix}$; **(d)** $\begin{bmatrix}4&8&1\\0&2&-2\\3&6&7\end{bmatrix}$.

2. 用 Gram-Schmidt 正交化, 求下列矩阵的 QR 分解.

(a) $\begin{bmatrix}2&3\\-2&-6\\1&0\end{bmatrix}$; **(b)** $\begin{bmatrix}-4&-4\\-2&7\\4&-5\end{bmatrix}$.

3. 用 Householder 反射, 求习题 1 中矩阵的 QR 分解.

4. 用 Householder 反射, 求习题 2 中矩阵的 QR 分解.

5. 用习题 2 或习题 4 中的 QR 分解, 求解最小二乘问题:

$$\text{(a)} \begin{bmatrix} 2 & 3 \\ -2 & -6 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -3 \\ 6 \end{bmatrix}; \quad \text{(b)} \begin{bmatrix} -4 & -4 \\ -2 & 7 \\ 4 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 9 \\ 0 \end{bmatrix}.$$

6. 求 QR 分解, 并用它来解最小二乘问题:

$$\text{(a)} \begin{bmatrix} 1 & 4 \\ -1 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \\ -3 \end{bmatrix}; \quad \text{(b)} \begin{bmatrix} 2 & 4 \\ 0 & -1 \\ 2 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ 2 \\ 1 \end{bmatrix}.$$

7. 证明: 方阵是正交的当且仅当它的各列是两两正交的单位向量.

8. 证明: 两个正交 $n \times n$ 矩阵之积仍是正交的.

9. 证明: $n \times n$ 矩阵的 Gram-Schmidt 正交化大约需要 n^3 次乘法以及 n^3 次加法.

10. 证明: 用于 QR 分解的 Householder 反射大约需要 $(2/3)n^3$ 次乘法以及 $(2/3)n^3$ 次加法.

11. 设 P 是 (4.27) 中定义的矩阵, 证明:

(a) $P^2 = P$; (b) P 是对称的; (c) $Pv = v$.

12. 证明 Householder 反射是对称矩阵.

计算机问题 4.3

1. 写出采用 Householder 反射实现 QR 分解的程序. 通过与 MATLAB 的qr 命令或等价的命令进行比较检查你的工作.

2. 用 QR 分解求以下不相容方程组的最小二乘解以及 2- 范数误差:

$$\text{(a)} \begin{bmatrix} 3 & -1 & 2 \\ 4 & 1 & 0 \\ -3 & 2 & 1 \\ 1 & 1 & 5 \\ -2 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ -5 \\ 15 \\ 0 \end{bmatrix};$$

$$\text{(b)} \begin{bmatrix} 4 & 2 & 3 & 0 \\ -2 & 3 & -1 & 1 \\ 1 & 3 & -4 & 2 \\ 1 & 0 & 1 & -1 \\ 3 & 1 & 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 2 \\ 0 \\ 5 \end{bmatrix}.$$

3. 设 A 是由 10×10 Hilbert 矩阵的最前 n 列组成的 $10 \times n$ 矩阵. 设 c 是 n 维向量 $[1, \cdots, 1]$, 并设 $b = Ac$. 用 QR 分解求解最小二乘问题 $Ax = b$, 其中 (a) $n = 6$, (b) $n = 8$, 并且与准确的最小二乘解 $\bar{x} = c$ 进行比较. 能计算到多少位准确的小数? 见计算

机问题 4.18, 在那里使用了正规方程.

4. 设 $x_1,\cdots,x_n$ 是 [2, 4] 中 11 个等距点, $y_i=1+x_i+x_i^2+\cdots+x_i^6$. 用 $\boldsymbol{QR}$ 分解计算最佳 d 次多项式, 这里 (a) $d=5$, (b) $d=6$, (c) $d=8$. 与习题 4.5 和计算机问题 4.1.9 进行比较. 系数能被计算到多少位准确小数?

4.4　非线性最小二乘

线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的最小二乘解把残差的 2- 范数 $\|\boldsymbol{Ax}-\boldsymbol{b}\|_2$ 极小化. 我们已学习了求这种解 $\bar{\boldsymbol{x}}$ 的两种方法: 一种基于正规方程, 另一种基于 $\boldsymbol{QR}$ 分解.

如果方程是非线性的, 那么上述两种方法都不能使用. 本节将介绍解非线性最小二乘问题的 Gauss-Newton 方法. 除了说明应用这种方法求解圆周相交问题之外, 还应用 Gauss-Newton 方法对数据进行具有非线性系数的模型的拟合.

4.4.1　Gauss-Newton 方法

考虑含 n 个未知量、m 个方程的方程组

$$\left\{\begin{aligned} r_1(x_1,\cdots,x_n)&=0,\\ &\vdots\\ r_m(x_1,\cdots,x_n)&=0.\end{aligned}\right. \tag{4.29}$$

用下面的函数

$$\boldsymbol{E}(x_1,\cdots,x_n)=\frac{1}{2}(r_1^2+\cdots+r_m^2)=\frac{1}{2}\boldsymbol{r}^{\mathrm{T}}\boldsymbol{r}$$

表示误差的平方和, 这里 $\boldsymbol{r}=[r_1,\cdots,r_m]^{\mathrm{T}}$. 在定义中已包括常数 $\frac{1}{2}$ 以简化后面的公式. 为了极小化 $\boldsymbol{E}$, 我们令梯度 $\boldsymbol{F}(x)=\nabla\boldsymbol{E}(x)$ 为零:

$$0=\boldsymbol{F}(\boldsymbol{x})=\nabla\boldsymbol{E}(x)=\nabla\left(\frac{1}{2}\boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{r}(\boldsymbol{x})\right)=\boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}}\mathrm{D}\boldsymbol{r}(\boldsymbol{x}). \tag{4.30}$$

注意到我们已经用了梯度的内积法则 (见附录 A).

我们通过回忆多元 Newton 方法开始, 为此需要向量值函数 $\boldsymbol{F}(x)=\boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}}\mathrm{D}\boldsymbol{r}(\boldsymbol{x})$ 的 Jacobi 矩阵. 把行向量写成列向量 $(\boldsymbol{r}^{\mathrm{T}}\mathrm{D}\boldsymbol{r})^{\mathrm{T}}=(\mathrm{D}\boldsymbol{r})^{\mathrm{T}}\boldsymbol{r}$. 可以用矩阵/向量乘积法则 (见附录 A) 得到

$$\mathrm{D}\boldsymbol{F}(\boldsymbol{x})=\mathrm{D}((\mathrm{D}\boldsymbol{r})^{\mathrm{T}}\boldsymbol{r})=(\mathrm{D}\boldsymbol{r})^{\mathrm{T}}\cdot\mathrm{D}\boldsymbol{r}+\sum_{i=1}^{m}r_i\mathrm{D}c_i,$$

这里 c_i 是 $\mathrm{D}\boldsymbol{r}$ 的第 i 列. 注意 $\mathrm{D}c_i=\boldsymbol{H}_{r_i}$($r_i$ 的二阶偏导数矩阵或者 r_i 的**Hessian**矩

阵),

$$\boldsymbol{H}_{r_i} = \begin{bmatrix} \dfrac{\partial^2 r_i}{\partial x_1 \partial x_1} & \cdots & \dfrac{\partial^2 r_i}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial^2 r_i}{\partial x_n \partial x_1} & \cdots & \dfrac{\partial^2 r_i}{\partial x_n \partial x_n} \end{bmatrix}.$$

如果省略求和项, 那么应用 Newton 方法就大大地被简化. 一种改进的 Newton 方法称为**拟 Newton**方法, 不直接使用函数的导数, 而是近似地把函数的导数置为零. 在求和中省略高阶项就给出以下拟 Newton 方法.

Gauss-Newton 方法

要极小化

$$r_1(\boldsymbol{x})^2 + \cdots + r_m(\boldsymbol{x})^2.$$

设 $\boldsymbol{x}^0 =$ 初始向量,

for $k = 0, 1, 2, \cdots$

$$\begin{aligned} \mathrm{D}\boldsymbol{r}(\boldsymbol{x}^k)^{\mathrm{T}} \mathrm{D}\boldsymbol{r}(\boldsymbol{x}^k) \boldsymbol{v}^k &= -\mathrm{D}\boldsymbol{r}(\boldsymbol{x}^k)^{\mathrm{T}} \boldsymbol{r}(\boldsymbol{x}^k) \\ \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \boldsymbol{v}^k \end{aligned} \tag{4.31}$$

end

注意, Gauss-Newton 方法的每一步使人回想起正规方程, 正规方程中的系数矩阵已被 $\mathrm{D}\boldsymbol{r}$ 代替. Gauss-Newton 方法求解误差平方的梯度的根. 尽管在极小值处梯度必须是零, 但反之不一定正确. 所以这种方法可能收敛于最大值或中性点 (neutral point). 在说明这种算法结果时必须小心.

两个相交圆相交于一点或两点, 除非这两个圆重合. 然而, 平面上的三个圆一般没有公共的交点. 在这样的情形下, 我们能够求平面上的点, 使它在最小二乘意义下与交点最近.

例 4.19 考虑平面上中心为 $(x_1, y_1) = (-1, 0)$, $(x_2, y_2) = (1, \frac{1}{2})$, $(x_3, y_3) = (1, -\frac{1}{2})$, 半径分别为 $R_1 = 1, R_2 = \frac{1}{2}, R_3 = \frac{1}{2}$ 的 3 个圆. 用 Gauss-Newton 方法求得 3 个圆的距离平方和最小的点.

3 个圆如图 4-11a 所示. 所求的点 (x, y) 使残差的平方和最小:

$$r_1(x, y) = \sqrt{(x - x_1)^2 + (y - y_1)^2} - R_1,$$

$$r_2(x, y) = \sqrt{(x - x_2)^2 + (y - y_2)^2} - R_2,$$

$$r_3(x, y) = \sqrt{(x - x_3)^2 + (y - y_3)^2} - R_3.$$

这是根据点(x,y)到以(x_1,y_1)为中心、R_1为半径的圆的距离是$|\sqrt{(x-x_1)^2+(y-y_1)^2}-R_1|$ 这个事实 (见习题 3). $\boldsymbol{r}(x,y)$ 的 Jacobi 矩阵是

$$\mathrm{D}\boldsymbol{r}(x,y)=\begin{bmatrix} \frac{x-x_1}{s_1} & \frac{y-y_1}{s_1} \\ \frac{x-x_2}{s_2} & \frac{y-y_2}{s_2} \\ \frac{x-x_3}{s_3} & \frac{y-y_3}{s_3} \end{bmatrix},$$

这里 $S_i=\sqrt{(x-x_i)^2+(y-y_i)^2}$ $(i=1,2,3)$. 初始向量 $(x^0,y^0)=(0,0)$ 的 Gauss-Newton 迭代收敛于 $(\bar{x},\bar{y})=(0.412\,891,0)$, 在 7 步后有 6 位准确小数. ◀

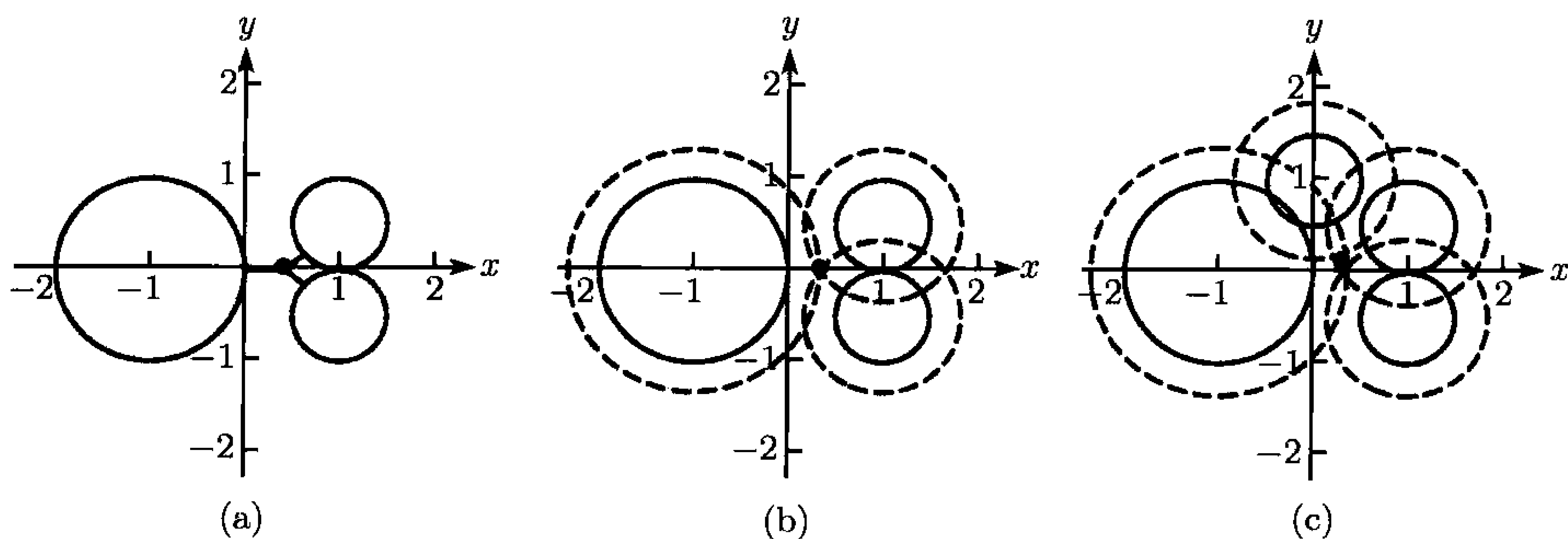


图 4-11　3 个圆的拟交点 (near-intersection point): (a) 用 Gauss-Newton 方法求得的最小二乘拟交点; (b) 用同一数量来延长 3 个圆的半径给出由多元 Newton 方法得出的不同类型的拟交点; (c) 例 4.21 中带有最小二乘解的点的 4 个圆

一个与 3 个圆有关的问题给出了不同类型的回答. 不是求与交点最相似的点, 我们可以用同一数量延长 (或收缩) 圆的半径, 直到它们有共同交点. 这等价于解方程组:

$$\begin{cases} r_1(x,y)=\sqrt{(x-x_1)^2+(y-y_1)^2}-(R_1+K)=0, \\ r_2(x,y)=\sqrt{(x-x_2)^2+(y-y_2)^2}-(R_2+K)=0, \\ r_3(x,y)=\sqrt{(x-x_3)^2+(y-y_3)^2}-(R_3+K)=0. \end{cases} \tag{4.32}$$

用这种方法确定的点 (x,y) 不一定与例 4.19 的最小二乘解相同.

例 4.20　采用例 4.19 中的圆, 通过解方程组 (4.32) 求 (x,y,K).

方程组包括含 3 个未知量的 3 个非线性方程, 需要用多元 Newton 方法. Jacobi 矩阵是

$$\mathrm{D}\boldsymbol{r}(x,y)=\begin{bmatrix} \frac{x-x_1}{s_1} & \frac{y-y_1}{s_1} & -1 \\ \frac{x-x_2}{s_2} & \frac{y-y_2}{s_2} & -1 \\ \frac{x-x_3}{s_3} & \frac{y-y_3}{s_3} & -1 \end{bmatrix}.$$

Newton 方法在 3 步内得到解 $(x, y, K) = (1/3, 0, 1/3)$. 交点 $(1/3, 0)$ 以及用 $K = 1/3$ 伸长半径的 3 个圆, 见图 4-11b. ◀

例 4.19 和例 4.20 就一组圆的 "拟交点" 表明了两种不同观点. 例 4.21 把这种方法结合在一起.

例 4.21 考虑圆心分别是 $(-1, 0), (1, 1/2), (1, -1/2), (0, 1)$, 半径分别是 $1, 1/2, 1/2, 1/2$ 的 4 个圆. 求点 (x, y) 以及常数 K, 使得这一点到半径增加了 K(于是分别为 $1 + K, 1/2 + K, 1/2 + K, 1/2 + K$) 的 4 个圆的距离的平方和成为极小.

该例直接把前面两个例题结合在一起. 有含 3 个未知量 x, y, K 的 4 个方程. 最小二乘残差类似于 (4.32), 但是带有 4 项, 并且 Jacobi 矩阵是

$$\mathrm{D}\boldsymbol{r}(x, y) = \begin{bmatrix} \frac{x-x_1}{s_1} & \frac{y-y_1}{s_1} - 1 \\ \frac{x-x_2}{s_2} & \frac{y-y_2}{s_2} - 1 \\ \frac{x-x_3}{s_3} & \frac{y-y_3}{s_3} - 1 \\ \frac{x-x_4}{s_4} & \frac{y-y_4}{s_4} - 1 \end{bmatrix}.$$

Gauss-Newton 方法提供了解 $(\bar{x}, \bar{y}) = (0.311\,385, 0.112\,268)$ 及 $\bar{K} = 0.367\,164$, 如图 4-11c 所示.

例 4.21 对三维球的模拟建立了全球定位系统 (GPS) 的数学基础. 见实例检验 4.

4.4.2 带非线性系数的模型

Gauss-Newton 方法的一种重要应用是对系数为非线性的模型进行拟合. 设 $(t_1, y_1), \cdots, (t_m, y_m)$ 是数据点, $y = f_{\boldsymbol{c}}(\boldsymbol{x})$ 是要拟合的函数, 这里 $\boldsymbol{c} = [c_1, \cdots, c_p]$ 是一组参数, 选择它们来极小化残差

$$\begin{cases} r_1(\boldsymbol{c}) = f_{\boldsymbol{c}}(t_1) - y_1 \\ \qquad\vdots \\ r_m(\boldsymbol{c}) = f_{\boldsymbol{c}}(t_m) - y_m \end{cases}$$

的平方和. 通常我们认为在这里足以有必要对 (4.29) 这种特殊情形作特别处理.

如果参数 $c_1, \cdots, c_p$ 以线性方式进入模型, 那么这是一组关于 c_i 的线性方程, 因此正规方程, 或者 QR 分解的解给出了参数 c 的最优选择. 如果参数 c_i 在模型中是非线性的, 那么进行同样处理会得到关于 c_i 是非线性的方程组的结果. 例如, 对数据点 (t_i, y_i) 拟合模型 $y = c_1 t^{c_2}$, 产生非线性方程组

$$\begin{cases} y_1 = c_1 t_1^{c_2}, \\ y_2 = c_1 t_2^{c_2}, \\ \quad\vdots \\ y_m = c_1 t_m^{c_2}. \end{cases}$$

因为 c_2 非线性地进入模型, 所以方程组不能写成矩阵形式.

亮点　收敛性

最小二乘问题中的非线性引起额外的挑战. 只要系数矩阵 $\boldsymbol{A}$ 满秩, 正规方程及 $\boldsymbol{QR}$ 分解方法就会求得唯一解. 另一方面, 用于非线性问题的 Gauss-Newton 迭代可能收敛于最小二乘误差的几个不同的相对极小值中的一个. 如果存在的话, 则对初始向量采用合理的近似方法有助于收敛到绝对极小值.

在 4.2 节中, 我们通过改变问题来处理这种困难. 通过对模型两边取对数 "使模型线性化", 而且通过最小二乘来极小化在这些对数变换坐标中的误差. 在对数变换坐标确实是可以极小化误差的情形下, 这种方法才是合适的.

然而, 为了解原来的最小二乘问题, 我们转向 Gauss-Newton 方法. 它用于对作为参数 $\boldsymbol{c}$ 的向量函数的误差函数 E 进行极小化. 矩阵 $\mathrm{D}\boldsymbol{r}$ 是误差 r_i 关于参数 c_j 的偏导数的矩阵, 它们是

$$(\mathrm{D}\boldsymbol{r})_{ij} = \frac{\partial r_i}{\partial c_j} = f_{c_i}(t_j).$$

有了这些信息, 就可以实现 Gauss-Newton 方法 (4.31) 了.

例 4.22　用 Gauss-Newton 方法, 把例 4.8 中的世界汽车供应量数据与 (非线性) 指数型模型进行拟合.

求数据对于指数型模型的最佳最小二乘拟合就是求 c_1, c_2, 使误差 $r_i = y_i - c_1 e^{c_2 t_i}$ $(i = 1, \cdots, m)$ 的 RMSE 极小化. 利用 4.4.1 节中的模型线性化, 我们极小化对数模型的误差 $\ln y_i - (\ln c_1 + c_2 t_i)$ 的 RMSE. 在两种不同意义下极小化 RMSE 的 c_i 值一般是不同的.

要通过 Gauss-Newton 方法计算最佳最小二乘拟合, 定义

$$\boldsymbol{r} = \begin{bmatrix} c_1 e^{c_2 t_1} - y_1 \\ \vdots \\ c_1 e^{c_2 t_m} - y_m \end{bmatrix},$$

并且对参数 c_1 及 c_2 取导数, 得到

$$\mathrm{D}\boldsymbol{r} = -\begin{bmatrix} \mathrm{e}^{c_2 t_1} & c_1 t_1 \mathrm{e}^{c_2 t_1} \\ \vdots & \vdots \\ \mathrm{e}^{c_2 t_m} & c_1 t_m \mathrm{e}^{c_2 t_m} \end{bmatrix}.$$

这个模型是要拟合世界汽车供应量数据, 这里 t 按年度量, 从 1970 年起, 汽车数量以 10^6 计. 进行 5 步 Gauss-Newton 方法 (4.31), 从初始估计 $(c_1, c_2) = (50, 0.1)$ 开始产生 $(c_1, c_2) \approx (58.51, 0.057\ 72)$, 它有 4 位精确数字. 对此数据的最佳最小二乘指数型模型是

$$y = 58.51\mathrm{e}^{0.057\ 72t}, \tag{4.33}$$

如图 4-12 所示. RMSE 是 7.68, 意思是在最小二乘意义下 7.68×10^6 辆汽车的平均模型误差.

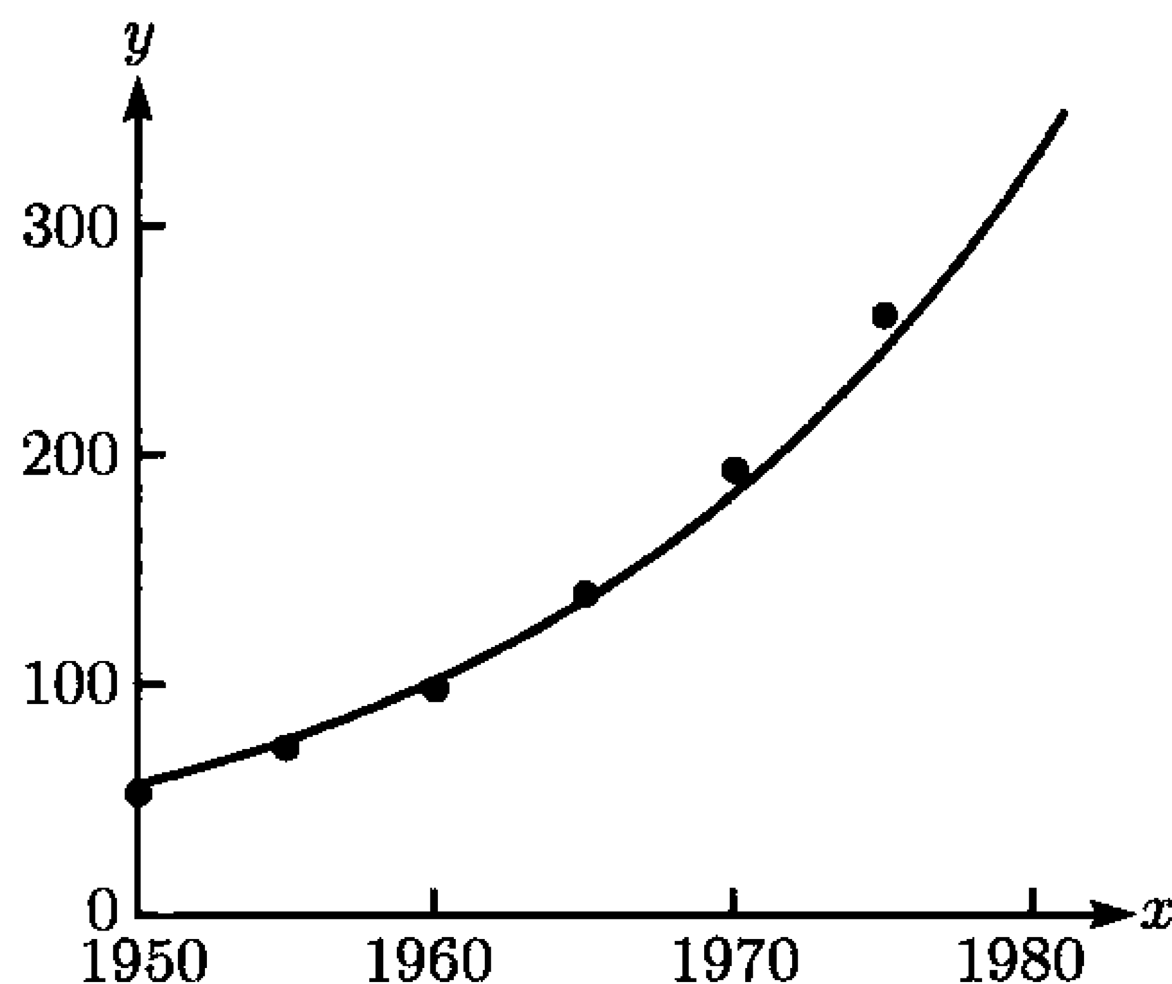


图 4-12 不使用线性化的世界汽车供应量数据的指数型拟合：最佳最小二乘拟合是$y = 58.51\mathrm{e}^{0.057\ 72t}$

最佳模型 (4.33) 可以与在例 4.8 中算得的最佳线性化指数型模型

$$y = 54.03\mathrm{e}^{0.061\ 52t}$$

进行比较. 这是从用于线性模型 $\ln y = \ln c_1 + c_2 t$ 的正规方程中得到的. 线性化模型的误差 r_i 的 RMSE 是 9.56, 比所要求的 (4.33) 的 RMSE 大. 然而, 线性化模型极小化误差 $\ln y_i - (\ln c_1 + c_2 t_i)$ 的 RMSE, 给出值 0.035 7, 比所要求的模型 (4.33) 相应的值 0.056 8 要低. 每一个模型在各自的数据空间是最优拟合.

原则上对解任何一个问题都有几种计算算法. 极小化 r_i 是标准的最小二乘问题, 但是使用者必须在数据背景的基础上确定极小化误差和极小化对数误差哪一个更合适. ◀

习题 4.4

1. Gauss-Newton 方法可用于求到 3 个圆的距离平方和最小的点 $(\bar{x},\bar{y})$. 取初始向量 $(x_0, y_0)=(0,0)$, 执行第一步求 (x_1,y_1). (a) 圆心是 (0, 1), (1, 1), (0, −1), 半径全是 1. (b) 圆心是 (−1, 0), (1, 1), (1, −1), 半径全是 1（计算机问题 1 要求 $(\bar{x},\bar{y})$）.
2. 对习题 1 中的 3 个圆执行应用到方程组 (4.32) 的多元 Newton 方法的第一步. 取 $(x_0,y_0, K_0)=(0,0,0)$. (计算机问题 2 要求解 (x,y,K).)
3. 证明: 点 (x,y) 到圆 $(x-x_1)^2+(y-y_1)^2=R_1^2$ 的距离是 $|\sqrt{(x-x_1)^2+(y-y_1)^2}-R_1|$.
4. 证明: 用于解线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的 Gauss-Newton 方法一步收敛到正规方程的解.
5. 求把 Gauss-Newton 迭代用于数据点为 $(t_1,y_1),(t_2,y_2),(t_3,y_3)$ 的模型拟合问题所需要的矩阵 $\mathrm{D}\boldsymbol{r}$. (a) 幂律模型 $y=c_1t^{c_2}$, (b) $y=c_1te^{c_2t}$.
6. 求把 Gauss-Newton 迭代用于数据点 $(t_1,y_1),(t_2,y_2),(t_3,y_3)$ 的模型拟合问题所需要的矩阵 $\mathrm{D}\boldsymbol{r}$. (a) 平移指数型模型 $y=c_3+c_1e^{c_2t}$ (b) 平移幂律模型 $y=c_3+c_1t^{c_2}$.
7. 证明 (4.32) 的实数解 (x,y,K) 的个数是无穷多个或者最多两个.

计算机问题 4.4

1. 用 Gauss-Newton 方法求到 3 个圆的距离平方和为极小的点 $(\bar{x},\bar{y})$. 使用初始向量 $(x_0, y_0)=(0,0)$. (a) 圆心是 (0, 1), (1, 1), (0, −1), 半径都是 1.(b) 圆心是 (−1, 0), (1, 1), (1, −1), 半径都是 1.
2. 对于计算机问题 1 中的 3 个圆, 把多元 Newton 方法用于方程组 (4.32). 使用初始向量 $(x_0,y_0,K_0)=(0,0,0)$.
3. 求点 (x,y) 及距离 K, 使得如同例 4.21 那样, 当圆半径增加 K 时, 该点到 4 个圆的距离平方和为最小. (a) 圆心是 (−1, 0), (1, 0), (0, 1), (0, −2), 半径都是 1. (b) 圆心是 (−2, 0), (3, 0), (0, 2), (0, −2) 半径都是 1.
4. 对下面的圆, 执行计算机问题 3 的步骤, 并且画出结果. (a) 圆心是 (−2, 0), (2, 0), (0, 2), (0, −2), (2, 2), 半径分别是 1, 1, 1, 1, 2. (b) 圆心是 (1, 1), (1, −1), (−1, 1), (2, 0), 半径都是 1.
5. 对例 4.10 中的身高/体重数据, 不经过线性化, 用 Gauss-Newton 方法拟合幂律模型. 计算 RMSE.
6. 不经过线性化, 对例 4.11 中的数据拟合血液浓度模型 (4.21).

实例检验 4　GPS、条件作用和非线性最小二乘

全球定位系统 (GPS) 由携带原子钟的 24 颗卫星组成, 它们沿离地球高度为 20 200km 的轨道运行. 6 个平面的每一平面中有 4 颗卫星, 平面相对于地极倾斜 55°, 每天作两次公转. 在任何时间, 从地球上任何点, 都可以直接看到 5~8 颗卫星. 每颗卫星有简单的任务: 仔细地从空间中预先确定的位置传送同步信号, 它们被地球上的 GPS 接收器所收集. 接收器利用这些信息进行数学处理 (稍后将会阐述), 确定接收器的精确坐标 (x,y,z).

在给定的瞬时, 接收器从第 i 个卫星收集到同步信号并且确定它的传递时间 t_i, 即发送信号与收到信号的时间差. 信号的标准速度是光速 $c\approx 299\ 792.458$ km/s. 用 c 乘传递时间给出接收器到卫星的距离. 把接收器放在以卫星的位置为中心、半径为 ct_i 的球的表面. 如果 3 颗

卫星都可用, 那么我们知道 3 个球面, 它们的交点由两个点组成, 如图 4-13 所示. 一个交点是接收器的位置, 另一个一般远离地球表面, 因此可以放心地忽略不计. 在理论上, 该问题化为计算这一交点, 即 3 个球面方程的公共解.

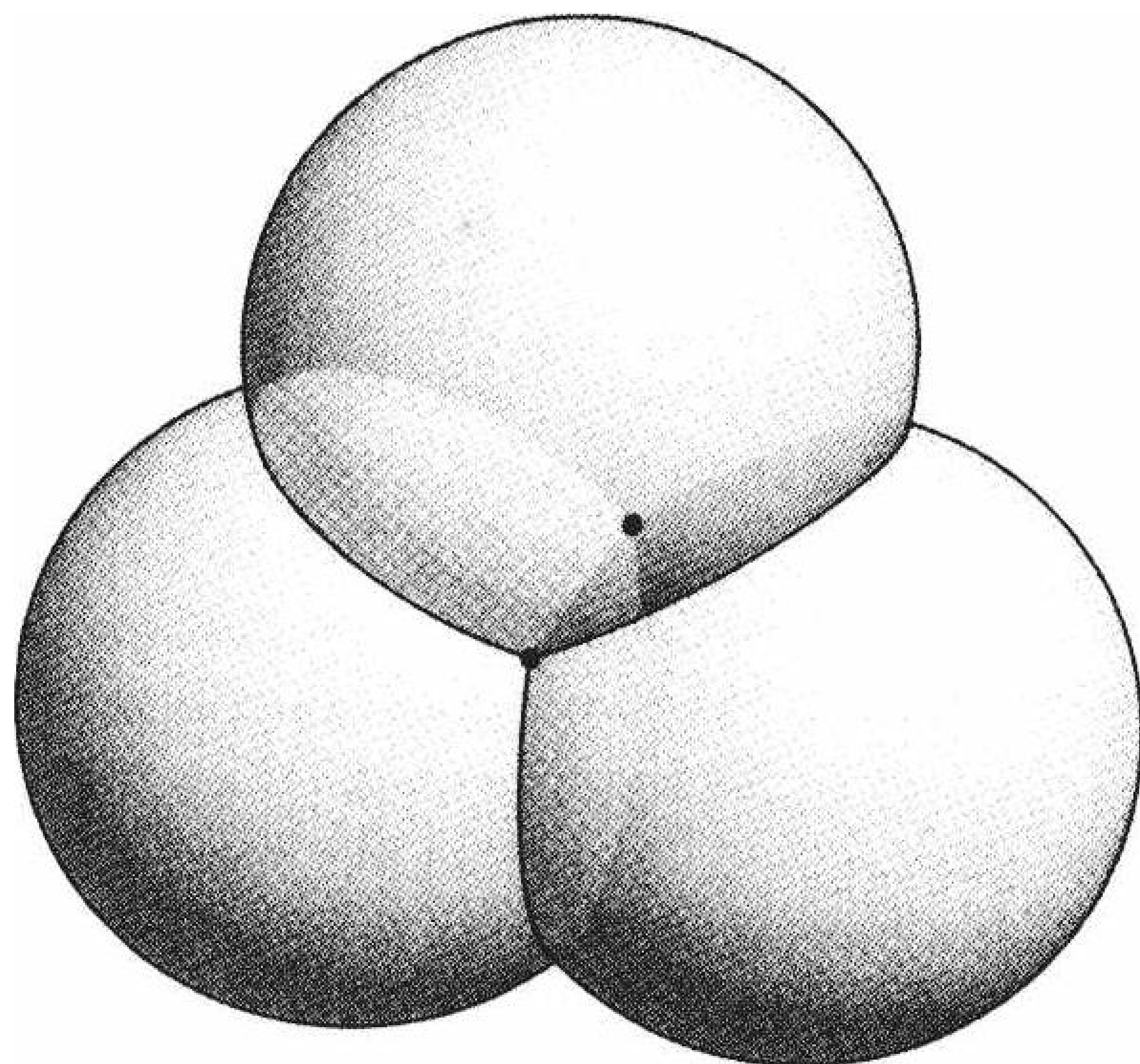


图 4-13 3 个相交的球. 一般地, 只有两点落在所有 3 个球上

然而, 前面的分析中有一个主要问题. 首先, 尽管卫星的信号传递时间通过它上面的原子钟进行的计时, 几乎精确到纳秒 (10^{-9} 秒), 但是地球上一般低成本的钟, 相对而言精确度较差. 如果我们求解方程时使用稍不精确的时间测定, 那么计算得到的位置可能有几千米的误差. 幸运的是, 有一种方法可以解决这个问题, 但需要使用一颗额外的卫星. 定义 d 是 (现在是 4 颗) 卫星上的钟与地球上接收器的钟的同步时间的差. 用 (A_i, B_i, C_i) 表示卫星 i 的位置. 那么真正的交点 (x, y, z) 满足

$$\begin{cases} (x-A_1)^2+(y-B_1)^2+(z-C_1)^2=[c(t_1-d)]^2, \\ (x-A_2)^2+(y-B_2)^2+(z-C_2)^2=[c(t_2-d)]^2, \\ (x-A_3)^2+(y-B_3)^2+(z-C_3)^2=[c(t_3-d)]^2, \\ (x-A_4)^2+(y-B_4)^2+(z-C_4)^2=[c(t_4-d)]^2, \end{cases} \tag{4.34}$$

求解未知量 x, y, z, d 即可. 求解这个方程组不仅得出接收器的位置, 而且由于 d, 也可以得出卫星上钟的正确时间. 因此, 通过使用一颗额外的卫星, 就能解决 GPS 接收器的钟的不精确性的问题.

从几何上来讲,4 个球可能没有交点, 但是, 如果把半径伸长或缩短一个合适的相同的量, 那么它们将有交点. 表示 4 个球相交的方程组 (4.34) 是表示平面中 3 个圆交点的 (4.32) 的三维模拟.

求解方程组 (4.34) 得到 (x, y, z, d) 并不困难. 注意, 从第一个方程减去后面 3 个方程就得到 3 个线性方程. 每个线性方程可以用于消去一个变量 x, y, z, 并且通过代入到任何一个原方程, 就会得到单个变量 d 的二次方程. 因此, 方程组 (4.34) 最多有两个实数解, 而且可以通过二次方程公式求得它们.

在应用 GPS 时进一步出现了两个问题. 首先是方程组 (4.34) 的条件作用. 我们将发现当卫星在天空中紧密集聚时, 求解 (x, y, z, d) 是病态的.

第二个困难是信号的传送速度并不精确地是 c. 信号要通过 100km 电离层和 10km 对流层, 它们的电磁性质可能影响传送速度. 更有甚者, 在到达接收器之前, 信号可能遇到地球上的障碍物, 即称为多路径干扰的影响. 如果认为这些障碍物在每个卫星的轨道上有相同的影响, 那么在 (4.34) 的右边引入时间校正 d 会有帮助. 然而, 这种假设通常是行不通的, 而且将引导我们从更多的卫星增加信息, 以及考虑用 Gauss-Newton 方法来求解最小二乘问题.

考虑原点是地球中心 (半径 $\approx$ 6370km) 的三维坐标系. GPS 接收器把这些坐标转变成纬度、经度和高度等数据, 供用全球信息系统 (GIS) 读取和用于更复杂的映射. 我们在这里不考虑其过程.

建议习题

1. 编写一个 MATLAB程序, 来求解前面叙述的二次求根公式. 对于已知的同步卫星位置 (15 600, 7 540, 10 380), (11 760, 2 750, 16 190), (11 610, 14 630, 7 680), (15 170, 610, 13 320) 单位是千米, 并且测量得到的时间间隔分别是 0.059 320 0, 0.051 920 0, 0.062 420 0, 0.055 710 0s, 求接收器靠近地球的位置 (x, y, z) 以及时间校正 d. 作为检查, 答案是 $(x, y, z) = (-19.053, 11.318, 6\ 370.252)$ 以及 $d = -0.000\ 021\ 26$s.

2. 如果有 MATLAB Symbolic Toolbox(或像 Maple 或 Mathematica 这种符号包) 可能替换第一步. 通过用`syms` 命令定义符号变量, 并用 Symbolic Toolbox 中的命令`solve` 解同步方程. 利用`subs` 把符号结果计算成浮点数.

3. 作为第一步的另一种替换, 通过使用多元 Newton 方法解方程组 (4.34). 令初始向量 $(x_0, y_0, z_0, d_0) = (0, 0, 6\ 370, 0)$. 把你的结果与第一步或者第二步比较.

现在建立 GPS 问题的条件作用的试验. 用球坐标 (ρ, ϕ_i, θ_i) 把卫星的位置 (A_i, B_i, C_i) 定义为

$$
\begin{aligned}
A_i &= \rho \cos \phi_i \cos \theta_i, \\
B_i &= \rho \cos \phi_i \sin \theta_i, \\
C_i &= \rho \sin \phi_i,
\end{aligned}
$$

其中 $\rho = 20\ 200$km 是固定的, 而 $0 \leqslant \phi_i \leqslant \pi/2$ 及 $0 \leqslant \theta_i \leqslant 2\pi$ $(i = 1, \cdots, 4)$ 是任选的. 限制 ϕ 坐标使得 4 个卫星在上半球. 设 $x = 0, y = 0, z = 6\ 370, d = 0.000\ 1$, 并且计算相应的卫星范围 $R_i = \sqrt{A_i^2 + B_i^2 + (C_i - 6\ 370)^2}$ 以及移动时间 $t_i = d + R_i/c$.

我们将定义特定于这种情形的误差放大因子. 卫星上的原子钟准确到大约 10ns 即 10^{-8}s. 因此, 研究这样大小的传送时间的改变的影响是重要的. 设后向或者输入误差是输入以米为单位的改变. 对于光速, $\triangle t_i = 10^{-8}$s, 相应于 $10^{-8}c \approx 3$m, 设前向或者输出误差是通过 t_i 这种改变造成的位置的改变 $\|(\triangle x, \triangle y, \triangle z)\|_\infty$, 也以米为单位. 那么我们能够定义无量纲

$$
\text{误差放大因子} = \frac{\|(\triangle x, \triangle y, \triangle z)\|_\infty}{c\|(\triangle t_1, \cdots, \triangle t_m)\|_\infty},
$$

以及问题的条件数是对一切小的 $\triangle t_i$ (譬如说 10^{-8} 或更小) 的最大的误差放大因子.

4. 改变之前由 $\triangle t_i = +10^{-8}$ 或 -10^{-8} 定义的每一个 t_i, 不必都相同. 现在用 $(\bar{x}, \bar{y}, \bar{z}, \bar{d})$ 表示方程 (4.34) 的新的解. 并且计算位置的差 $\|(\triangle x, \triangle y, \triangle z)\|_\infty$ 以及误差放大因子. 试

用 $\triangle t_i$ 的不同的变化, 以米为单位, 求出最大的位置误差. 以算出的误差放大因子为基础, 估计这个问题的条件数.

5. 现在用更紧密组合的一组卫星重复第 4 步. 选择彼此在 5% 以内的 ϕ_i 以及 5% 以内的 θ_i. 如在第 4 步那样带或者不带相同的输入误差进行求解. 求出最大的位置误差以及误差放大因子. 当卫星紧密地或松散地聚在一起时, 比较 GPS 问题的条件作用.

6. 确定是否可以通过增加卫星来减小 GPS 误差及条件数. 回到第 4 步中没有密集配置卫星的情形, 再增加 4 颗卫星. (在所有时间及地球上的每一位置, 能看见 5~12 颗 GPS 卫星.) 设计 Gauss-Newton 迭代来求解 4 个变量 (x, y, z, d)、8 个方程的最小二乘方程组. 好的初始向量是什么? 求出最大的 GPS 位置误差, 并且估计条件数. 从 4 颗不密集配置、4 颗密集配置以及 8 颗不密集配置的卫星, 总结你的结果, 哪种配置形式最佳? 以米为单位, 最大的 GPS 误差是什么? 这些是根据卫星信号你完全应该预期到的.

软件和进一步阅读

最小二乘近似起始于 19 世纪早期. 和多项式插值一样, 可以把它看成一种有损耗的压缩形式, 用于对复杂的或者有噪声的数据集求出简单的表达式. 直线函数、多项式、指数型函数以及幂律函数是实通常采用的模型. 周期数据需要三角表达式, 它们被取到极值就导出第 10 章中介绍的三角插值和三角最小二乘拟合.

通过使用 4.2 节的 3 步方法, 我们能够用系数是线性的任何函数去拟合数据, 结果是正规方程的解. 对于病态问题, 不建议用正规方程, 这是因为这种方法的条件数大致按平方增大. 在这种情形下优先选用的矩阵分解方法是 $\boldsymbol{QR}$ 分解, 而在某些情形下是第 12 章介绍的奇异值分解. 教科书 [3] 是关于 $\boldsymbol{QR}$ 和其他矩阵分解的优秀参考书, 教科书 [5] 全面介绍了最小二乘的基础知识. 更加专业的参考书 [1,2,6] 覆盖了最小二乘拟合线性及多元回归的统计学知识.

如果方程组相容, 用 MATLAB的反斜线命令对于 $\boldsymbol{Ax} = \boldsymbol{b}$ 的执行 Gauss 消去法; 如果方程组不相容, 通过 $\boldsymbol{QR}$ 分解求解最小二乘问题. MATLAB的`qr` 命令是以 LAPACK 的程序 DGEQRF 为基础的. IMSL 库为最小二乘数据拟合提供了程序 RLINE. NAG 库程序 E02ADF 对多项式执行最小二乘近似方法, 如同 MATLAB的`polyfit` 一样. 诸如 S^+, SAS, SPSS, 以及 Minitab 这类统计学软件包执行各种回归分析.

非线性最小二乘涉及拟合在模型中的非线性系数. 对于这种计算 Gauss-Newton 方法是首选工具, 尽管收敛性没有保证, 而且甚至在收敛时, 也不意味着有唯一的最佳值. 我们可以在 MATLAB Optimization Toolbox (MATLAB 最优化工具箱) 中找到关于 Gauss-Newton 方法的软件. 关于 GPS 的数学知识的介绍见 [7], 关于这个专题的一般信息见 [4].

第 5 章　数值微分和数值积分

计算机辅助制造取决于对沿规定路径运动的精确控制. 数控车床和铣床依靠通常由计算机辅助设计软件给出的三次或 Bézier 样条这种参数曲线画出切割或刨削工具的轨迹. 计算机生成的动画制作、计算机游戏以及虚拟现实的应用都面临着类似的问题.

实例检验　5.5 节后的实例检验探讨了沿着任意的参数路径的速度控制问题. 对于按规定速度在曲线上移动的路径参数, 这条曲线关于弧长被重新参数化. 对弧长的积分应用自适应求积法就提供了取得这种控制的有效方法.

微积分计算的主要问题是计算函数的导数和积分. 对这种问题可以采取两种方法: 数值计算和符号计算. 本章将讨论这两个问题, 但是将十分详细地研究数值计算问题. 导数和积分都有明确的数学定义, 但是用户想要的答案类型常常与函数定义的方式有关.

像 $f(x) = \sin x$ 这一类函数的导数是微积分引论所研究的问题. 如果函数用基本函数给出, 比如 $f(x) = \sin^3(x^{\tan x} \cosh x)$, 它的三阶导数通过符号计算方法可以较快地求得, 这里的微积分法则是由计算机来执行的. 在答案能用基本函数表达的情形中, 对反导数而言相同的结论也是正确的.

在实践中, 对一个已知的函数还可以用另外两种常用的方法. 一个函数可以定义为一个表格. 例如由实验测量得到的时间/温度对表格 $\{(t_1, T_1), \cdots (t_n, T_n)\}$(或许是在相等间隔的时间). 在这种情形下, 用大学一年级微积分中的法则来求导数和反导数是不可能的. 最后, 函数可以定义为实验或计算机模拟的输出, 而它们的输入是由用户所确定. 在后两种情形中, 不能使用符号计算方法, 解决这种问题需要用数值微分和积分.

5.1　数 值 微 分

首先, 我们建立用于近似导数的有限差分公式. 在某些情形下, 这就是计算的目的. 在第 7 章和第 8 章中, 将用这些公式离散常微分方程和偏微分方程.

5.1.1　有限差分公式

根据定义,$f(x)$ 在 x 处的导数是

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \tag{5.1}$$

当然, 前提是这个极限存在. 这就为近似在 x 处的导数导出了一个有用的公式. Taylor 定理告诉我们, 如果 f 是二次连续可微的, 那么

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(c), \tag{5.2}$$

这里 c 在 x 和 $x+h$ 之间. 等式 (5.2) 意味着以下公式:

两点前向差分公式

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(c), \tag{5.3}$$

这里 c 在 x 和 $x+h$ 之间.

在有限计算中, 我们在 (5.1) 中不能取极限, 但是如果 h 是小量, (5.3) 意味着商式非常接近这个导数. 我们通过计算近似式

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{5.4}$$

并把 (5.3) 中的最后一项处理为误差, 来使用 (5.3). 因为由近似所造成的误差正比于增量 h, 所以我们可以通过使 h 变小而使误差变小. 两点前向差分公式是近似一阶导数的一阶方法. 一般地, 如果误差是 $O(h^n)$, 我们就称公式是 ***n* 阶**近似.

称这个公式 "一阶" 的微妙之处是 c 与 h 有关. 一阶的概念是当 $h \to 0$ 时, 误差应正比于 h. 当 $h \to 0$ 时,c 是移动目标, 因此比例常数改变了. 但是只要 f'' 连续, 当 $h \to 0$ 时比例常数 $f''(c)$ 趋于 $f''(x)$, 这就使称公式是一阶的是恰当的.

亮点　收敛性

两点前向差分方法的误差公式 $-hf''(c)/2$ 的好处是什么? 我们在试图近似 $f'(x)$, 因此我们很可能不知道 $f''(x)$. 有两种回答. 首先, 在查对代码和软件时, 一种好的检验是把它运行到已经完全解决的例子上. 这里的正确答案是知道的, 因此能把误差与预期的进行比较. 在这种情形下, 我们能够知道 $f''(x)$ 就像知道 $f'(x)$ 一样. 其次, 即使我们不能求出整个公式的值, 了解误差关于 h 的尺度常常是有益的. 公式是一阶这个事实意味着当 h 减半时误差应该近似地减半 (即使我们无法计算比例常数 $f''(c)/2$).

例 5.1　取 $h = 0.1$, 用两点前向差分公式近似 $f(x) = 1/x$ 在 $x = 2$ 处的导数. 对两点前向差分公式 (5.4) 求值得到

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} = \frac{\frac{1}{2.1} - \frac{1}{2}}{0.1} \approx -0.238\,1.$$

这个近似值与导数 $f'(x) = -x^{-2}$ 在 $x = 2$ 处的正确值的差就是误差

$$-0.238\,1 - (-0.250\,0) = 0.011\,9.$$

把这个结果与由公式 $hf''(c)/2$(c 在 2 和 2.1 之间) 预测的误差进行比较. 由于 $f''(x) = 2x^{-3}$, 误差一定在下面两个数之间:

$$(0.1)2^{-3} \approx 0.012\,5 \quad 和 \quad (0.1)(2.1)^{-3} \approx 0.010\,8.$$

这与我们的结果是一致的. 然而这个信息通常是得不到的. ◀

用更高级的策略可以建立二阶公式. 根据 Taylor 定理, 如果 f 是三次连续可微, 那么

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(c_1),$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(c_2).$$

这里 $x - h < c_2 < x < c_1 < x + h$. 把这两个等式相减得到以下带有显式误差项的三点公式:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}f'''(c_1) - \frac{h^2}{12}f'''(c_2). \tag{5.5}$$

为了使新的公式中的误差项更精确, 我们将用下面的定理:

定理 5.1 (一般中值定理)　设 f 是区间 $[a,b]$ 上的连续函数, $x_1, \cdots, x_n$ 是 $[a,b]$ 中的点, 而且 $a_1, \cdots, a_n > 0$, 那么在 a, b 之间存在数 c 使得

$$(a_1 + \cdots + a_n)f(c) = a_1 f(x_1) + \cdots + a_n f(x_n). \tag{5.6}$$

证　设这 n 个函数值中最小的一个是 $f(x_i)$, 最大值的一个是 $f(x_j)$, 那么

$$a_1 f(x_i) + \cdots + a_n f(x_i) \leqslant a_1 f(x_1) + \cdots + a_n f(x_n) \leqslant a_1 f(x_j) + \cdots + a_n f(x_j).$$

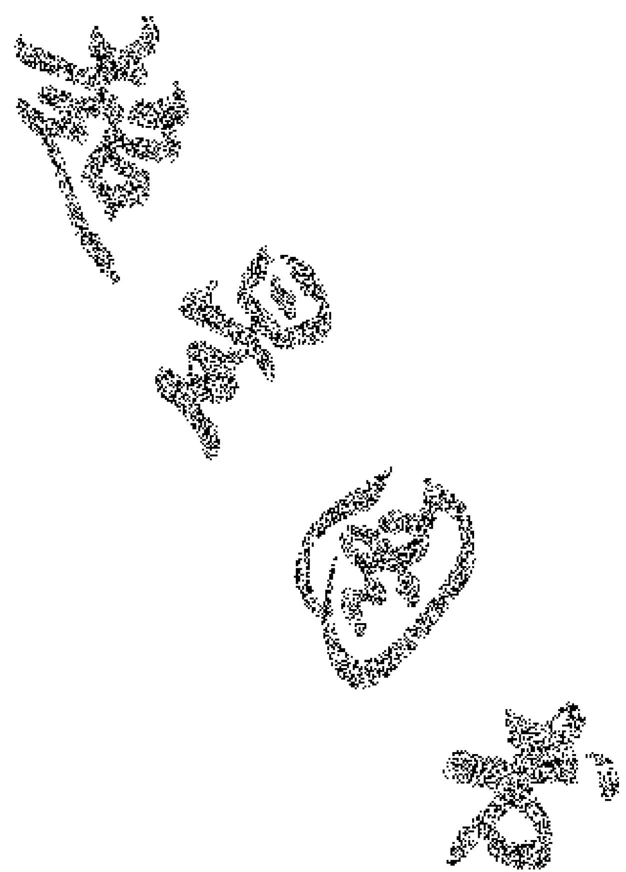
即

$$f(x_i) \leqslant \frac{a_1 f(x_1) + \cdots + a_n f(x_n)}{a_1 + \cdots + a_n} \leqslant f(x_j).$$

根据中值定理, 在 x_i 和 x_j 之间存在常数 c, 使得

$$f(c) = \frac{a_1 f(x_1) + \cdots + a_n f(x_n)}{a_1 + \cdots + a_n}.$$

因此,(5.6) 式成立.



定理 5.1 告诉我们：可以把 (5.5) 中的最后两项结合起来，得到一个二阶公式：

三点中心差分公式

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(c), \tag{5.7}$$

这里 $x - h < c < x + h$.

例 5.2 取 $h = 0.1$, 用三点中心差分公式近似 $f(x) = \frac{1}{x}$ 在 $x = 2$ 处的导数.

用三点中心差分公式求值得到

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} = \frac{\frac{1}{2.1} - \frac{1}{1.9}}{0.2} \approx -0.250\,6.$$

误差是 0.000 6, 比例 5.1 中的两点前向差分公式的误差有了改进. ◀

亮点 收敛性

当 $h \to 0$ 时, 两点和三点近似公式都收敛于导数 (尽管以不同的速度). 这两个公式都由于很接近的数相减从而破坏了浮点运算的基本规则. 但是因为求导是一种固有的不稳定过程, 所以不可能得到改善. 对于非常小的 h 值, 舍入误差将影响计算, 如例 5.3 所示.

用同样的方法可以得到高阶导数的近似公式. 例如, Taylor 展开式

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(c_1),$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(c_2).$$

这里 $x - h < c_2 < x < c_1 < x + h$, 把它们相加消去一阶导数项得到

$$f(x+h) + f(x-h) - 2f(x) = h^2 f''(x) + \frac{h^4}{24} f^{(4)}(c_1) + \frac{h^4}{24} f^{(4)}(c_2).$$

利用定理 5.1 把误差项结合起来并且两边都除 h^2 就得到以下公式：

二阶导数的三点中心差分公式

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} + \frac{h^2}{12} f^{(4)}(c) \tag{5.8}$$

对 $x - h$ 和 $x + h$ 之间的某个 c 成立.

5.1.2 舍入误差

到目前为止, 本章的所有公式都破坏了第 0 章中的规则, 即建议不要进行很接近的数的相减. 这对于数值微分是一个极大的困难, 但是它在本质上是不可能避免的. 为了把问题理解得更清楚, 考虑以下例子:

例 5.3 求 $f(x)=\mathrm{e}^x$ 在 $x=0$ 处的导数的近似.

两点公式 (5.4) 给出

$$f'(x)\approx\frac{\mathrm{e}^{x+h}-\mathrm{e}^x}{h},\tag{5.9}$$

三点公式 (5.7) 给出

$$f'(x)\approx\frac{\mathrm{e}^{x+h}-\mathrm{e}^{x-h}}{2h}.\tag{5.10}$$

对 $x=0$ 及增量 h 的很广的范围, 这些公式的结果以及与正确的导数值 $\mathrm{e}^0=1$ 比较而得到的误差在表 5-1 中给出.

表 5-1

h	公式 (5.9)	误 差	公式 (5.10)	误 差
10^{-1}	1.051 709 180 756 48	−0.051 709 180 756 48	1.001 667 500 198 44	−0.001 667 500 198 44
10^{-2}	1.005 016 708 416 79	−0.005 016 708 416 79	1.000 016 666 749 99	−0.000 016 666 749 99
10^{-3}	1.000 500 166 708 38	−0.000 500 166 708 38	1.000 000 166 666 68	−0.000 000 166 666 68
10^{-4}	1.000 050 001 667 14	−0.000 050 001 667 14	1.000 000 001 666 89	−0.000 000 001 666 89
10^{-5}	1.000 005 000 006 96	−0.000 005 000 006 96	1.000 000 000 012 10	−0.000 000 000 012 10
10^{-6}	1.000 000 499 962 18	−0.000 000 499 962 18	0.999 999 999 973 24	0.000 000 000 026 76
10^{-7}	1.000 000 049 433 68	−0.000 000 049 433 68	0.999 999 999 473 64	0.000 000 000 526 36
10^{-8}	0.999 999 993 922 53	0.000 000 006 077 47	0.999 999 993 922 53	0.000 000 006 077 47
10^{-9}	1.000 000 082 740 37	−0.000 000 082 740 37	1.000 000 027 229 22	−0.000 000 027 229 22

开始时候, 当 h 减小时误差减小, 对两点前向差分公式 (5.4) 和三点中心差分公式 (5.7) 分别得到很接近预期的误差 $O(h)$ 和 $O(h^2)$. 但是, 注意, 当 h 再进一步减小时近似式反而变坏了.

对于非常小的 h, 近似式失去精度的原因是有效数字的损失. 这两个公式都有很相近数相减, 损失了有效数字, 而且由于被很小的数相除而扩大了这种影响, 从而使结果更糟. ◀

为了更好地理解数值微分公式对损失有效数字的敏感性, 我们详细地分析三点中心差分公式. 用 $\hat{f}(x+h)$ 表示输入 $f(x+h)$ 的浮点形式, $\hat{f}(x+h)$ 与正确值 $f(x+h)$ 相差一个机器 ε 阶的数量. 对于目前的讨论, 我们将假定函数值是一阶, 所以相对误差和绝对误差大约相等.

因为 $\hat{f}(x+h)=f(x+h)+\varepsilon_1$, $\hat{f}(x-h)=f(x-h)+\varepsilon_2$, 这里 $|\varepsilon_1|,|\varepsilon_2|\approx\varepsilon_{\text{mach}}$, 准确的 $f'(x)$ 和三点中心差分公式 (5.7) 的机器形式之差是

$$\begin{aligned}
f'(x)_{\text{准确}}-f'(x)_{\text{机器}} &= f'(x)-\frac{\hat{f}(x+h)-\hat{f}(x-h)}{2h}\\
&= f'(x)-\frac{f(x+h)+\varepsilon_1-(f(x-h)+\varepsilon_2)}{2h}\\
&= \left(f'(x)-\frac{f(x+h)-f(x-h)}{2h}\right)+\frac{\varepsilon_2-\varepsilon_1}{2h}\\
&= \left(f'(x)_{\text{准确}}-f'(x)_{\text{公式}}\right)+\text{误差}_{\text{舍入}}.
\end{aligned}$$

我们可以认为全体误差是截断误差 (准确导数和准确近似公式之间的差以及舍入误差之和), 它度量了计算机实现的公式损失有效数字的大小. 舍入误差的绝对值满足

$$\left|\frac{\varepsilon_2-\varepsilon_1}{2h}\right|\leqslant\frac{2\varepsilon_{\text{mach}}}{2h}=\frac{\varepsilon_{\text{mach}}}{h},$$

这里 $\varepsilon_{\text{mach}}$ 表示机器 ε. 因此 $f'(x)$ 机器近似的误差的绝对值的上界是

$$E(h)\equiv\frac{h^2}{6}f'''(c)+\frac{\varepsilon_{\text{mach}}}{h},\tag{5.11}$$

这里 $x-h<c<x+h$. 以前我们仅考虑了误差的第一项, 即数学上的误差. 表 5-1 迫使我们还要考虑有效数字的损失.

画出如图 5-1 所示的函数 $E(h)$ 是有用的. $E(h)$ 的最小值出现在方程

$$0=E'(h)=-\frac{\varepsilon_{\text{mach}}}{h^2}+\frac{M}{3}h\tag{5.12}$$

的解处, 这里我们已用 M 来近似 $|f'''(c)|\approx|f'''(x)|$. 解 (5.12) 得到

$$h=(3\varepsilon_{\text{mach}}/M)^{\frac{1}{3}}.$$

它对增量 h 给出了最小的整体误差, 包括了计算机舍入. 在双精度制中, 它近似于 $\varepsilon_{\text{mach}}^{\frac{1}{3}}\approx10^{-5}$, 这与表 5-1 相一致.

上面得到的主要结论是: 当 h 减小直到成为机器 ε 的立方根时, 三点差分公式在精度上将得到改进; 而当 h 小于机器 ε 的立方根时, 误差可能再次开始增大.

对于其他公式, 关于舍入误差的分析能够得到类似的结果. 习题 16 要求读者对两点前向差分公式分析舍入误差的影响.

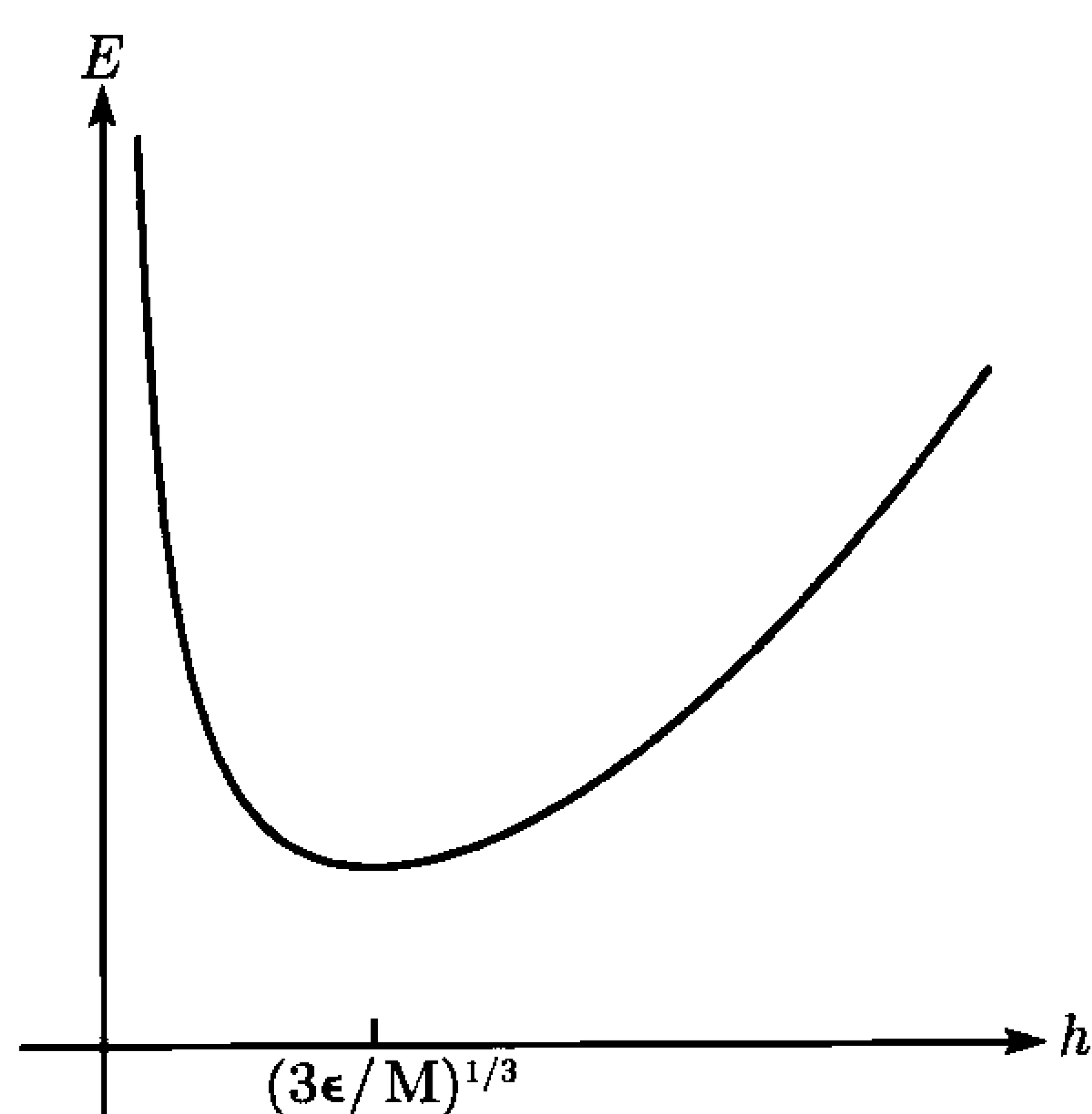


图 5-1　舍入误差对数值微分的影响: 对小的 h, 误差是以舍入误差为主

5.1.3　外推

假设我们提出一个 n 阶公式 $F(n)$ 来近似给定的量 Q. 这里阶的意思是

$$Q \approx F(h) + Kh^n,$$

这里 K 在我们感兴趣的 h 范围内大致是一个常数. 相关的例子是

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \frac{f'''(c_h)}{6}h^2, \tag{5.13}$$

这里我们已经强调未知点 c_h 在 x 和 $x+h$ 之间, 但与 h 有关. 尽管 c_h 不是常数, 但如果函数 f 适当光滑, 而且 h 不太大, 那么误差系数 $f'''(c_h)/6$ 的值不应该离 $f'''(x)/6$ 太远.

在类似这种情形中, 可以用一些代数知识使一个 n 阶的公式提高一阶. 因为我们知道公式 $F(h)$ 的阶是 n, 所以如果取 $\frac{h}{2}$ 代替 h 来使用公式, 那么误差应该从 h^n 的常数倍减小到 $(\frac{h}{2})^n$ 的常数倍, 或者按因子 2^n 缩小. 换言之, 我们期望

$$Q - F\left(\frac{h}{2}\right) \approx \frac{1}{2^n}(Q - F(h)). \tag{5.14}$$

我们正是依赖 K 大致是常数的假设. 注意到从 (5.14) 容易解出问题中的量 Q, 它给出以下公式:

n 阶公式的外推

$$Q \approx \frac{2^n F(h/2) - F(h)}{2^n - 1}. \tag{5.15}$$

这就是对 $F(h)$ 的**外推**(extrapolation) 公式. **外推**有时也称为**Richardson 外推**, 一般对 Q 给出比 $F(h)$ 更高阶的近似. 为了要明白原因, 假设能把 n 阶公式

$F_n(h)$ 写成

$$Q = F_n(h) + Kh^n + O(h^{n+1}).$$

然后把 h 减半得到

$$Q = F_n\left(\frac{h}{2}\right) + K\frac{h^n}{2^n} + O(h^{n+1}),$$

而且, 外推形式 $F_{n+1}(h)$ 将满足

$$\begin{aligned}F_{n+1}(h) &= \frac{2^n F_n(\frac{h}{2}) - F_n(h)}{2^n - 1} \\ &= \frac{2^n(Q - K\frac{h^n}{2^n} - O(h^{n+1})) - (Q - Kh^n - O(h^{n+1}))}{2^n - 1} \\ &= Q + \frac{-Kh^n + Kh^n + O(h^{n+1})}{2^n - 1} = Q + O(h^{n+1}).\end{aligned}$$

因此, $F_{n+1}(h)$ 至少是近似量 Q 的 $n+1$ 阶公式.

例 5.4 对公式 (5.13) 用外推.

我们从对导数 $f'(x)$ 的二阶中心差分公式 $F_2(h)$ 开始. 外推公式 (5.15) 给出 $f'(x)$ 的一个新的公式:

$$\begin{aligned}F_4(x) &= \frac{2^2 F_2(\frac{h}{2}) - F_2(h)}{2^2 - 1} \\ &= \left[4\frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} - \frac{f(x+h) - f(x-h)}{2h}\right]/3 \qquad (5.16) \\ &= \frac{f(x-h) - 8f(x-\frac{h}{2}) + 8f(x+\frac{h}{2}) - f(x+h)}{6h}.\end{aligned}$$

这是一个五点中心差分公式. 前面的论证保证这个公式至少有 3 阶, 但是由于 3 阶误差项抵消了, 它结果就有 4 阶. 事实上, 由于经过检查, $F_4(h) = F_4(-h)$, 对于 h 的误差与对于 $-h$ 的误差相同. 因此误差项只可能是 h 的偶次幂. ◀

例 5.5 对二阶导数公式 (5.8) 用外推.

这个方法也是二阶的, 所以对 $n = 2$ 使用外推公式 (5.15). 外推公式是

$$\begin{aligned}F_4(x) &= \frac{2^2 F_2(\frac{h}{2}) - F_2(h)}{2^2 - 1} \\ &= \left[4\frac{f(x+\frac{h}{2}) - 2f(x) + f(x-\frac{h}{2})}{h^2/4} - \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}\right]/3 \\ &= \frac{-f(x-h) + 16f(x-\frac{h}{2}) - 30f(x) + 16f(x+\frac{h}{2}) - f(x+h)}{3h^2}.\end{aligned}$$

这种近似二阶导数的新方法是 4 阶的, 其原因同上. ◀

5.1.4　符号微分法和符号积分法

MATLAB符号工具箱包含获得符号形式的函数的符号导数的指令. 以下指令是直观的:

```
>> syms x;
>> f=sin(3*x);
>> f1=diff(f)

f1=

3*cos(3*x)

>>
```

3 阶导数也容易求得:

```
>>f3=diff(f,3)

f3=

-27*cos(3*x)
```

积分使用 MATLAB符号指令`int`:

```
>>syms x
>>f=sin(x)

f=

sin(x)

>>int(f)

ans=

-cos(x)

>>int(f,0,pi)

ans=

2
```

对更复杂的函数, 用 MATLAB指令`pretty`检验最后所得到的答案, 并且用指令`simple`把它简化, 它们都大有帮助, 如以下的代码:

```
>>syms x

>>f=sin(x)^7
```

```
f=

sin(x)^7

>>int(f)

ans=

-1/7*sin(x)^6*cos(x)-6/35*sin(x)^4*cos(x)-8/35*sin(x)^2*cos(x)
     -16/35*cos(x)
>>pretty(simple(int(f)))
                                     3              5              7
                 -cos(x) + cos(x)  - 3/5 cos(x)   + 1/7 cos(x)
```

当然,对于某些被积函数,不存在用初等函数表达的不定积分. 用函数 $f(x)=e^{\sin x}$ 试验,可以看到 MATLAB放弃了. 像在这种情形中一样,除了 5.2 节中的数值方法外,没有其他可能.

习题 5.1

1. 用两点前向差分公式近似 $f'(1)$,并求出近似误差,这里 $f(x)=\ln x$. **(a)** $h=0.1$; **(b)** $h=0.01$; **(c)** $h=0.001$.

2. 用三点中心差分公式近似 $f'(0)$,这里 $f(x)=e^x$. **(a)** $h=0.1$; **(b)** $h=0.01$; **(c)** $h=0.001$.

3. 用两点前向差分公式近似 $f'(\frac{\pi}{3})$,这里 $f(x)=\sin x$,并求出近似误差. 还要求出由误差项所蕴含的界,并证明近似误差位于上下界之间. **(a)** $h=0.1$; **(b)** $h=0.01$; **(c)** $h=0.001$.

4. 用三点中心差分公式执行习题 3 中的步骤.

5. 用对二阶导数的三点中心差分公式来近似 $f''(1)$,这里 $f(x)=x^{-1}$. **(a)** $h=0.1$; **(b)** $h=0.01$; **(c)** $h=0.001$. 求出近似误差.

6. 用对二阶导数的三点中心差分公式来近似 $f''(0)$,这里 $f(x)=\cos x$. **(a)** $h=0.1$; **(b)** $h=0.01$; **(c)** $h=0.001$. 求出近似误差.

7. 建立近似 $f'(x)$ 的包括误差项的两点后向差分公式.

8. 求近似公式 $f'(x)=\frac{4f(x+h)-3f(x)-f(x-2h)}{6h}$ 的误差项和阶.

9. 通过对两点前向差分公式进行外推,求近似 $f'(x)$ 的一种二阶公式.

10. **(a)** 对于 $f(x)=\frac{1}{x}$,计算两点前向差分公式对 $f'(x)$ 的近似,这里 x 和 h 任意;
(b) 减去准确答案即得到误差,证明它近似地正比于 h;
(c) 重做 (a) 和 (b),改用三点中心差分公式. 现在误差应该正比于 h^2.

11. 建立仅用数据 $f(x-h)$、$f(x)$ 及 $f(x+3h)$ 的二阶方法来近似 $f'(x)$,求出误差项.

12. **(a)** 对习题 11 中建立的公式进行外推;
(b) 通过近似 $f'(\frac{\pi}{3})$ 来说明新公式的阶,这里 $f(x)=\sin x$,取 $h=0.1$ 及 $h=0.01$.

13. 建立仅用数据 $f(x-h)$、$f(x)$ 及 $f(x+3h)$ 的一阶方法来近似 $f''(x)$,求出误差项.

14. **(a)** 对习题 13 中建立的公式进行外推得到 $f''(x)$ 的一个二阶公式;
(b) 通过近似 $f''(0)$ 来说明新公式的阶,这里 $f(x)=\cos x$,取 $h=0.1$ 及 $h=0.01$.

15. 建立仅用数据 $f(x-2h)$、$f(x)$ 及 $f(x+3h)$ 的二阶方法来近似 $f'(x)$,求出误差项.

16. 求 $E(h)$ 即一阶导数的两点前向差分公式的机器近似的误差上界, 效仿前面 (5.11) 的推理. 求出对应 $E(h)$ 最小值的 h.

17. 证明 3 阶导数的二阶公式:

$$f'''(x)=\frac{-f(x-2h)+2f(x-h)-2f(x+h)+f(x+2h)}{2h^3}+O(h^2).$$

18. 证明 3 阶导数的二阶公式:

$$f'''(x)=\frac{f(x-3h)-6f(x-2h)+12f(x-h)-10f(x)+3f(x+h)}{2h^3}+O(h^2).$$

19. 证明 4 阶导数的二阶公式:

$$f^{(4)}(x)=\frac{f(x-2h)-4f(x-h)+6f(x)-4f(x+h)+f(x+2h)}{h^4}+O(h^2).$$

这个公式用于实例检验 2.

20. 这个习题证明了实例检验 2 中的方程 (2.42) 及 (2.43). 设 $f(x)$ 是 5 次连续可微函数.

(a) 证明如果 $f(x)=f'(x)=0$, 那么

$$f^{(4)}(x)=\frac{12f(x+h)-6f(x+2h)+\frac{4}{3}f(x+3h)}{h^4}-\frac{6}{5}f^{(5)}(c)h;$$

(b) 证明如果 $f''(x)=f'''(x)=0$, 那么

$$f^{(4)}(x)=\frac{12f(x-3h)-24f(x-2h)+12f(x-h)}{25h^4}-\frac{18}{25}f^{(5)}(c)h;$$

(c) 证明如果 $f''(x)=f'''(x)=0$, 那么

$$f^{(4)}(x)=\frac{25f(x-4h)-93f(x-3h)+111f(x-2h)-43f(x-h)}{25h^4}+\frac{217}{100}f^{(5)}(c)h.$$

21. 利用 Taylor 展开式证明 (5.16) 是一个 4 阶公式.

22. $f'(x)$ 的两点前向差分公式中的误差项可以写成另一种方式. 证明另一种结果

$$f'(x)=\frac{f(x+h)-f(x)}{h}-\frac{h}{2}f''(x)-\frac{h^2}{6}f'''(c).$$

这里 c 在 x 与 $x+h$ 之间. 我们在第 8 章推导 Crank-Nicolson 方法时将用到这种误差形式.

23. 研究外推这个名称的由来. 假设 $F(h)$ 是近似一个量 Q 的 n 阶公式, 考虑 xy 平面中的点 $(Kh^2, F(h))$ 及 $(K(h/2)^n, F(h/2))$, 这里的误差画在 x 轴上而公式输出在 y 轴上, 求通过这两点的直线 (是误差与 F 之间关系的最佳函数近似). 当你把误差外推为 0 时, 这条直线在 y 轴上的截距就是公式的值. 证明这个外推值由公式 (5.15) 给出.

计算机问题 5.1

1. 对 $f'(0)$ 的三点中心差分公式的误差作一个表格, 这里 $f(x)=\sin x-\cos x$, 取 $h=10^{-1},\cdots,10^{-12}$, 如同 5.1.2 节中的表格. 绘出结果图, 最小误差与理论预期一致吗?

2. 如在计算机问题 1 中,$f(x)=x^{-1}$, 对 $f'(1)$ 的三点中心差分公式的误差作表并绘图.

3. 如在计算机问题 1 中, $f(x) = \sin x - \cos x$, 对 $f'(0)$ 的两点前向差分公式的误差作表并绘图. 把你的答案与习题 16 中建立的理论作比较.
4. 如在计算机问题 3 中, $f(x) = x^{-1}$, 近似 $f'(1)$ 作表并绘图. 把你的答案与习题 16 中建立的理论进行比较.
5. 如在计算机问题 1 中对 $f(x) = \cos x$ 近似 $f''(0)$, 用三点中心差分公式, 绘出图来. 误差的最小值在何处出现? 用机器 ε 来表示.

5.2 数值积分的 Newton-Cotes 公式

定积分的数值计算依赖于许多我们已经见过的相同工具. 在第 3 章和第 4 章, 我们利用插值和最小二乘模型建立关于一组数据点的函数近似的方法. 我们将讨论基于这两种思想的**数值积分**(numerical integration) **或求积分**(quadrature) 的方法.

例如, 给定一个定义在区间 $[a, b]$ 上的函数 f, 我们可以经过 $f(x)$ 的某些点作出其插值多项式. 因为多项式的定积分计算简单, 所以可以把这种计算用于近似 $f(x)$ 的积分. 这就是 Newton-Cotes 逼近近似积分. 另外, 我们还可以求在最小二乘意义下对函数近似得好的低次多项式, 并且用它的积分作为近似积分, 这种方法称为 Gauss 求积. 本章将讨论这两种方法.

为了建立 Newton-Cotes 公式, 我们需要 3 种简单的定积分, 如图 5-2 所示.

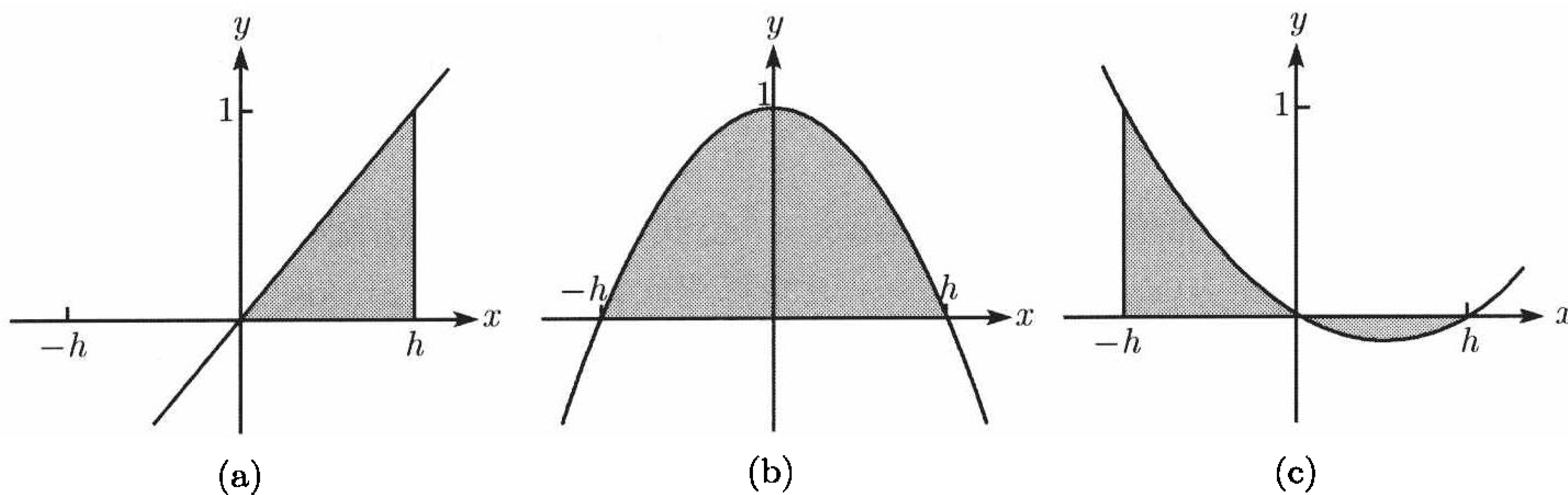


图 5-2 3 种简单定积分 (5.17)、(5.18) 及 (5.19): 有效面积是 (a) $\frac{h}{2}$, (b) $\frac{4h}{3}$, (c) $\frac{h}{3}$

图 5-2a 表示在插值数据点 $(0, 0)$ 和 $(h, 1)$ 的直线之下的面积. 其面积是高为 1、底为 h 的三角形的面积, 所以这个面积是

$$\int_0^h \frac{x}{h} \mathrm{d}x = \frac{h}{2}. \tag{5.17}$$

图 5-2b 表示在插值数据点 $(-h, 0)$、$(0, 1)$ 及 $(h, 0)$ 的抛物线之下的面积, 它有面积

$$\int_{-h}^{h} P(x) \mathrm{d}x = x - \frac{x^3}{3h^2} = \frac{4}{3}h. \tag{5.18}$$

图 5-2c 表示在 x 轴与插值数据点 $(-h,1)$、$(0,0)$ 及 $(h,0)$ 的抛物线之间的面积，具有有效面积

$$\int_{-h}^{h} P(x)\mathrm{d}x = \frac{1}{3}h. \tag{5.19}$$

5.2.1 梯形法则

我们从基于插值的数值积分的最简应用开始. 设 $f(x)$ 是定义在区间 $[x_0, x_1]$ 上的有连续二阶导数的函数, 如图 5-3a 所示. 用 $y_0 = f(x_0)$ 及 $y_1 = f(x_1)$ 表示相应的函数值. 考虑经过 (x_0, y_0) 及 (x_1, y_1) 的一次插值多项式 $P_1(x)$. 利用 Lagrange 公式, 求出带有误差项的插值多项式

$$f(x) = y_0\frac{x-x_1}{x_0-x_1} + y_1\frac{x-x_0}{x_1-x_0} + \frac{(x-x_0)(x-x_1)}{2!}f''(c_x) = P(x) + E(x).$$

可以证明 "未知点" c_x 连续依赖于 x.

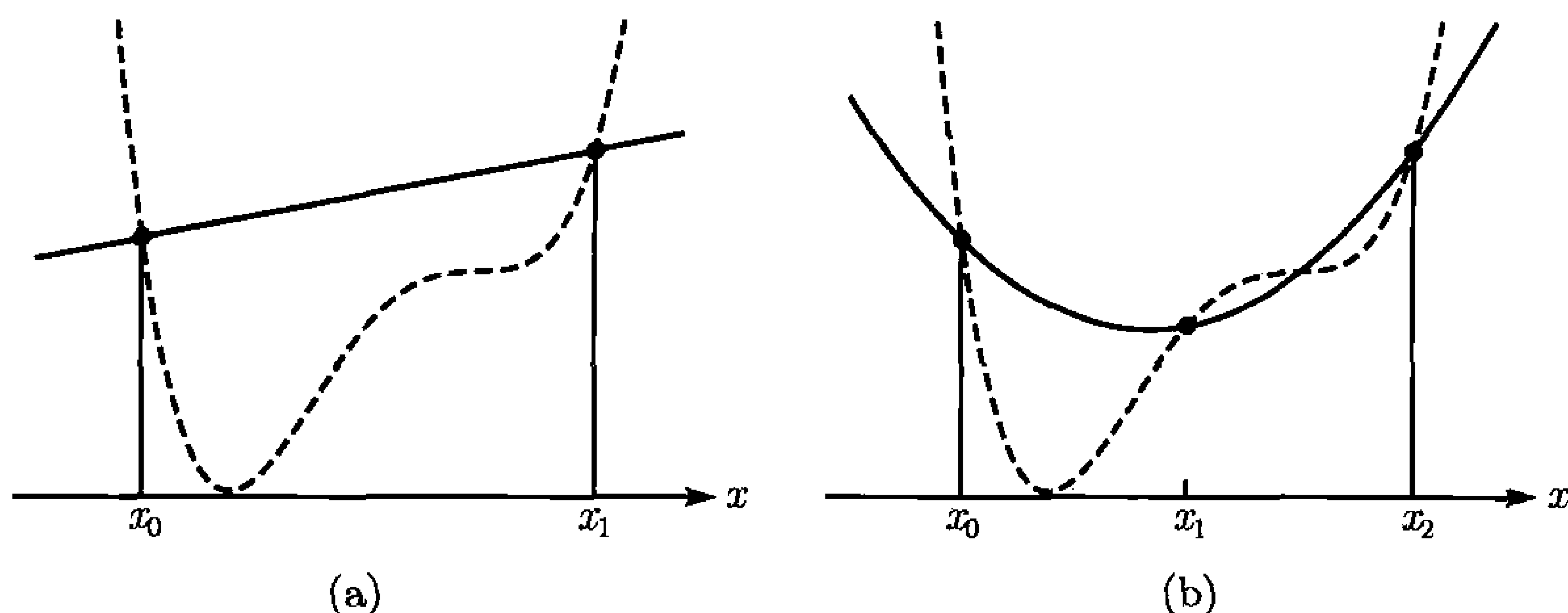


图 5-3 基于插值的 Newton-Cotes 公式: (a) 梯形法则用插值了 $(x_0, f(x_0))$ 与 $(x_1, f(x_1))$ 的直线代替函数; (b) Simpson 法则使用了在三点 $(x_0, f(x_0))$、$(x_1, f(x_1))$ 及 $(x_2, f(x_2))$ 的抛物线插值函数

上式两边在区间 $[x_0, x_1]$ 积分得

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x = \int_{x_0}^{x_1} P(x)\mathrm{d}x + \int_{x_0}^{x_1} E(x)\mathrm{d}x.$$

计算第一个积分给出

$$\int_{x_0}^{x_1} P(x)\mathrm{d}x = y_0\int_{x_0}^{x_1} \frac{x-x_1}{x_0-x_1}\mathrm{d}x + y_1\int_{x_0}^{x_1} \frac{x-x_0}{x_1-x_0}\mathrm{d}x = y_0\frac{h}{2} + y_1\frac{h}{2} = h\frac{y_0+y_1}{2}, \tag{5.20}$$

这里已经定义 $h = x_1 - x_0$ 是积分区间的长度, 并且利用 (5.17) 来计算积分. 例如在第一个积分中作代换 $w = -x + x_1$ 给出

$$\int_{x_0}^{x_1} \frac{x-x_1}{x_0-x_1}\mathrm{d}x = \int_{h}^{0} \frac{-w}{-h}(-\mathrm{d}w) = \int_{0}^{h} \frac{w}{h}\mathrm{d}w = \frac{h}{2},$$

而第二个积分是

$$\int_{x_0}^{x_1}\frac{x-x_0}{x_1-x_0}\mathrm{d}x=\int_0^h\frac{w}{h}\mathrm{d}w=\frac{h}{2}.$$

公式 (5.20) 计算了一个梯形的面积, 即给出了法则本身的名称.

误差项是

$$\int_{x_0}^{x_1}E(x)\mathrm{d}x=\frac{1}{2!}\int_{x_0}^{x_1}(x-x_0)(x-x_1)f''(c(x))\mathrm{d}x=\frac{f''(c)}{2}\int_{x_0}^{x_1}(x-x_0)(x-x_1)\mathrm{d}x$$

$$=\frac{f''(c)}{2}\int_0^h u(u-h)\mathrm{d}u=-\frac{h^3}{12}f''(c),$$

这里我们用了定理 0.9(积分中值定理). 我们证明了:

梯形法则

$$\int_{x_0}^{x_1}f(x)\mathrm{d}x=\frac{h}{2}(y_0+y_1)-\frac{h^3}{12}f''(c), \tag{5.21}$$

这里 c 在 x_0 与 x_1 之间.

5.2.2 Simpson 法则

图 5-3b 说明了类似于梯形法则的**Simpson 法则**, 只不过是用抛物线代替了一次插值多项式. 和前面一样, 我们可以把被积函数写成插值抛物线与插值误差之和:

$$\begin{aligned}f(x)&=y_0\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}+y_1\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\\&\quad+y_2\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}+\frac{(x-x_0)(x-x_1)(x-x_2)}{3!}f'''(c_x)\\&=P(x)+E(x).\end{aligned}$$

积分给出

$$\int_{x_0}^{x_2}f(x)\mathrm{d}x=\int_{x_0}^{x_2}P(x)\mathrm{d}x+\int_{x_0}^{x_2}E(x)\mathrm{d}x,$$

这里

$$\begin{aligned}\int_{x_0}^{x_2}P(x)\mathrm{d}x&=y_0\int_{x_0}^{x_2}\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\mathrm{d}x+y_1\int_{x_0}^{x_2}\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\mathrm{d}x\\&\quad+y_2\int_{x_0}^{x_2}\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\mathrm{d}x\\&=y_0\frac{h}{3}+y_1\frac{4h}{3}+y_2\frac{h}{3}.\end{aligned}$$

我们已设 $h = x_2 - x_1 = x_1 - x_0$, 而且用 (5.18) 求中间的积分, 而用 (5.19) 求第一个和第三个积分. 计算误差项的积分 (证明省略) 为

$$\int_{x_0}^{x_2} E(x)\mathrm{d}x = -\frac{h^5}{90}f^{(4)}(c).$$

只要 $f^{(4)}$ 存在而且连续, 上式就对区间 $[x_0, x_2]$ 中的某个 c 成立. 从而得到 Simpson 法则.

Simpson 法则

$$\int_{x_0}^{x_2} f(x)\mathrm{d}x = \frac{h}{3}(y_0 + 4y_1 + y_2) - \frac{h^5}{90}f^{(4)}(c). \tag{5.22}$$

这里 $h = x_2 - x_1 = x_1 - x_0$, c 在 x_0 和 x_2 之间.

例 5.6 用梯形法则和 Simpson 法则近似

$$\int_1^2 \ln x \, \mathrm{d}x,$$

并求出你的近似误差的上界.

用梯形法则计算

$$\int_1^2 \ln x\mathrm{d}x \approx \frac{h}{2}(y_0 + y_1) = \frac{1}{2}(\ln 1 + \ln 2) = \frac{\ln 2}{2} \approx 0.346\ 6.$$

梯形法则的误差是 $-h^3 f''(c)/12$, 这里 $1 < c < 2$. 因为 $f''(x) = -1/x^2$, 误差的值最大是

$$\frac{1^3}{12c^2} \leqslant \frac{1}{12} \approx 0.083\ 4.$$

换言之, 梯形法则告诉我们

$$\int_1^2 \ln x\mathrm{d}x = 0.346\ 6 \pm 0.083\ 4.$$

这个积分可以用分部积分精确地计算出来:

$$\int_1^2 \ln x\mathrm{d}x = x\ln x|_1^2 - \int_1^2 \mathrm{d}x = 2\ln 2 - 1\ln 1 - 1 \approx 0.386\ 294. \tag{5.23}$$

梯形法则的近似积分值和误差上界与这一结果相一致.

用 Simpson 法则计算得到

$$\int_1^2 \ln x \mathrm{d}x \approx \frac{h}{3}(y_0 + 4y_1 + y_2) = \frac{0.5}{3}(\ln 1 + 4\ln\frac{3}{2} + \ln 2) \approx 0.385\ 8.$$

Simpson 法则的误差是 $-h^5 f^{(4)}(c)/90$, 这里 $1 < c < 2$. 因为 $f^{(4)}(x) = -6/x^4$, 其误差最大是

$$\frac{6(0.5)^5}{90c^4} \leqslant \frac{6(0.5)^5}{90} = \frac{1}{480} \approx 0.002\ 1.$$

于是, Simpson 法则告诉我们

$$\int_1^2 \ln x \mathrm{d}x = 0.385\ 8 \pm 0.002\ 1,$$

它又与准确值相一致, 而且比梯形法则近似积分值更精确. ◀

对像梯形法则或者 Simpson 法则这样的数值积分法则进行比较的一种方法是通过比较误差项. 这一信息是简单地通过以下定义传达的:

定义 5.2 *若一种数值积分方法用于所有 k 次或小于 k 次的多项式的积分都是精确的, 那么这种最大的整数 k 就是这种数值积分方法的***精度**(degree of precision).

例如, 梯形法则的误差项 $-h^3 f''(c)/12$ 表明, 如果 $f(x)$ 是一次或小于一次的多项式, 那么误差将是零, 因此这种多项式将被精确地积分. 因此梯形法则的精度是 1. 这明显来自几何直观, 因为一条直线之下的面积用梯形法则近似是精确的.

不太明显的是, Simpson 法则的精度是 3, 但是这恰是 (5.22) 中的误差项所表明的. 这个意外结果的几何基础是这样的事实: 一条抛物线与一条 3 次曲线相交于 3 个等距点, 那么这条抛物线与这条 3 次曲线在这个区间有相同的积分 (习题 15).

例 5.7 求称为**Simpson 3/8 法则**的 3 次 Newton-Cotes 公式的精度:

$$\int_{x_0}^{x_3} f(x)\mathrm{d}x \approx \frac{3h}{8}(y_0 + 3y_1 + 3y_2 + y_3).$$

只要连续试验单项式就可以了. 我们将把细节留给读者. 例如, 当 $f(x) = x^2$ 时, 检查恒等式

$$\frac{3h}{8}[x^2 + 3(x+h)^2 + 3(x+2h)^2 + (x+3h)^2] = \frac{(x+3h)^3 - x^3}{3},$$

后者即为 x^2 在 $[x, x+3h]$ 上的精确积分. 对 $1, x, x^2, x^3$ 等式依然成立, 但是对 x^4 并不成立. 因此该法则的精度是 3. ◀

梯形法则及 Simpson 法则是 "闭"Newton-Cotes 公式的例子, 因为它们包括了被积函数在区间端点的求值. 开 Newton-Cotes 公式对不可能的情形 (例如, 当我们近似反常积分时) 是有用的. 5.2.4 节将讨论开公式.

5.2.3 复合 Newton-Cotes 公式

梯形法则与 Simpson 法则限制在单个区间上运算. 当然, 因为定积分在各个子区间上具有可加性, 所以可以通过把区间分成几个子区间进行积分, 在每一部分上分别应用这些法则, 然后全部加起来. 这种策略叫做**复合数值积分**.

复合梯形法则就是简单地对在相邻子区间或 “段” 上的梯形法则近似进行求和.

要近似计算

$$\int_a^b f(x)\mathrm{d}x,$$

考虑沿着 x 轴的等距网格点

$$a = x_0 < x_1 < x_2 < \cdots < x_{m-2} < x_{m-1} < x_m = b$$

这里 $h = x_{i+1} - x_i$ 对每一个 i 成立, 如图 5-4 所示. 在每一个子区间作出带有误差项的近似

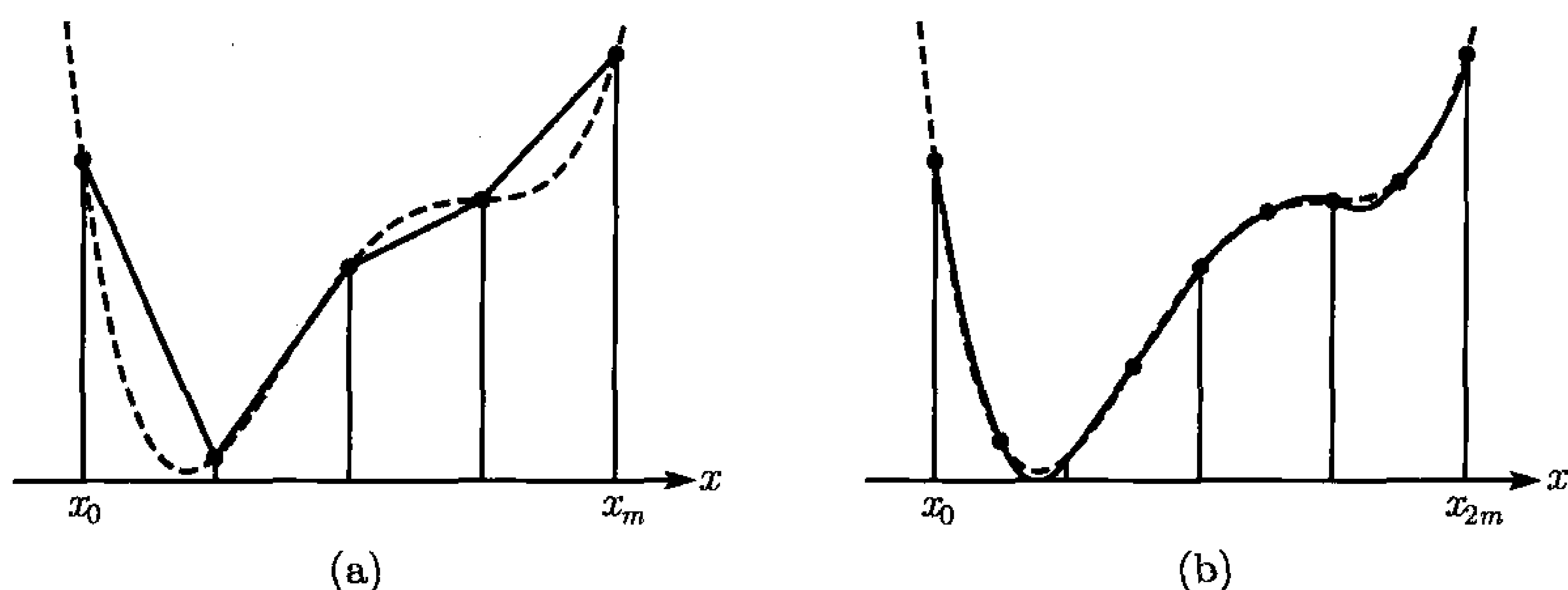


图 5-4 Newton-Cotes 复合公式: (a) 复合梯形法则把在 m 个相邻子区间上的梯形法则公式相加; (b) 复合 Simpson 法则与 Simpson 法则做法相同

$$\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x = \frac{h}{2}(f(x_i) + f(x_{i+1})) - \frac{h^3}{12}f''(c_i),$$

这里假设 f'' 是连续的. 对所有的子区间相加 (注意在内部子区间上函数值重叠) 得到

$$\int_a^b f(x)\mathrm{d}x = \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{m-1} f(x_i)\right] - \sum_{i=0}^{m-1}\frac{h^3}{12}f''(c_i).$$

按照定理 5.1, 对某个 $a < c < b$, 误差项可以写成

$$\frac{h^3}{12}\sum_{i=0}^{m-1} f''(c_i) = \frac{h^3}{12}mf''(c).$$

因为 $mh = (b-a)$, 所以误差项是 $(b-a)h^2f''(c)/12$. 总结一下, 如果 f'' 在 $[a,b]$ 上连续, 那么以下结论成立:

复合梯形法则

$$\int_a^b f(x)\mathrm{d}x = \frac{h}{2}\left(y_0 + y_m + 2\sum_{i=1}^{m-1} y_i\right) - \frac{(b-a)h^2}{12}f''(c) \tag{5.24}$$

这里 $h = (b-a)/m$, c 在 a 和 b 之间.

复合 Simpson 法则遵照相同的策略. 考虑 x 轴上的等距网格点

$$a = x_0 < x_1 < x_2 < \cdots < x_{2m-2} < x_{2m-1} < x_{2m} = b$$

这里 $h = x_{i+1} - x_i$ 对每个 i 成立. 在每一个长度 $2h$ 的区间 $[x_{2i}, x_{2i+2}](i = 0, \cdots, m-1)$ 上执行 Simpson 方法. 换言之, 被积函数 $f(x)$ 在每个子区间上被 x_{2i}, x_{2i+1} 及 x_{2i+2} 处拟合的插值抛物线所近似, 对它进行积分并求和. 在子区间上带有误差项的近似是

$$\int_{x_{2i}}^{x_{2i+2}} f(x)\mathrm{d}x = \frac{h}{3}\left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right] - \frac{h^5}{90}f^{(4)}(c_i),$$

这一次, 仅在编号为偶数的 x_j 处函数值重叠. 对所有子区间相加, 得到

$$\int_a^b f(x)\mathrm{d}x = \frac{h}{3}\left[f(a) + f(b) + 4\sum_{i=1}^{m} f(x_{2i-1}) + 2\sum_{i=1}^{m-1} f(x_{2i})\right] - \sum_{i=0}^{m-1}\frac{h^5}{90}f^{(4)}(c_i).$$

按照定理 5.1, 对某个 $a < c < b$, 可以把误差项写成

$$\frac{h^5}{90}\sum_{i=0}^{m-1} f^{(4)}(c_i) = \frac{h^5}{90}mf^{(4)}(c).$$

因为 $m \cdot 2h = (b-a)$, 所以误差项是 $(b-a)h^4f^{(4)}(c)/180$. 假设 $f^{(4)}$ 在 $[a,b]$ 上连续, 以下结论成立:

复合 Simpson 法则

$$\int_a^b f(x)\mathrm{d}x = \frac{h}{3}\left(y_0 + y_{2m} + 4\sum_{i=1}^{m} y_{2i-1} + 2\sum_{i=1}^{m-1} y_{2i}\right) - \frac{(b-a)h^4}{180}f^{(4)}(c), \tag{5.25}$$

这里 c 在 a 和 b 之间.

例 5.8 用复合梯形法则和复合 Simpson 法则对

$$\int_1^2 \ln x\mathrm{d}x$$

执行 4 段近似.

对于 [1, 2] 上的复合梯形法则, 4 段意味着 $h=\frac{1}{4}$. 近似式是

$$\int_1^2 \ln x \mathrm{d}x \approx \frac{\frac{1}{4}}{2}\left(y_0+y_4+2\sum_{i=1}^{3} y_i\right)=\frac{1}{8}\left[\ln 1+\ln 2+2\left(\ln\frac{5}{4}+\ln\frac{6}{4}+\ln\frac{7}{4}\right)\right]\approx 0.383\ 7.$$

其误差最多是

$$\frac{(b-a)h^2}{12}|f''(c)|=\frac{\frac{1}{16}}{12}\frac{1}{c^2}\leqslant\frac{1}{(16)(12)(1^2)}=\frac{1}{192}\approx 0.005\ 2.$$

4 段的 Simpson 法则取 $h=\frac{1}{8}$. 近似式是

$$\begin{aligned}\int_1^2 \ln x \mathrm{d}x &\approx \frac{\frac{1}{8}}{3}\left(y_0+y_8+4\sum_{i=1}^{4} y_{2i-1}+2\sum_{i=1}^{3} y_{2i}\right)\\ &=\frac{1}{24}\left[\ln 1+\ln 2+4\left(\ln\frac{9}{8}+\ln\frac{11}{8}+\ln\frac{13}{8}+\ln\frac{15}{8}\right)+2\left(\ln\frac{5}{4}+\ln\frac{6}{4}+\ln\frac{7}{4}\right)\right]\\ &\approx 0.386\ 292.\end{aligned}$$

这与从 (5.23) 得到的正确值 0.386 294 相比, 有 5 位小数是一致的. 事实上, 误差不可能大于

$$\frac{(b-a)h^4}{180}|f^{(4)}(c)|=\frac{(\frac{1}{8})^4}{180}\frac{6}{c^4}\leqslant\frac{6}{8^4\times 180\times 1^4}\approx 0.000\ 008.$$ ◀

例 5.9 使用复合 Simpson 法则求 $\int_0^{\pi}\sin^2 x \mathrm{d}x$ 的近似值, 为了得到 6 位正确小数, 求出必需的段数 m.

我们要求误差满足

$$\frac{(\pi-0)h^4}{180}|f^{(4)}(c)|<0.5\times 10^{-6}.$$

因为 $\sin^2 x$ 的 4 阶导数是 $-8\cos 2x$, 我们需要

$$\frac{\pi h^4}{180}\times 8<0.5\times 10^{-6},$$

或者 $h<0.043\ 5$, 因此 $m=\mathrm{ceil}(\pi/(2h))=37$ 段就足够了. ◀

5.2.4 开 Newton-Cotes 方法

像梯形法则和 Simpson 法则这些所谓闭 Newton-Cotes 方法需要积分区间端点的输入值. 在区间端点有可去奇点的某些被积函数用开 Newton-Cotes 方法处理较

为容易. 这种方法不用端点的值. 以下法则适用于二阶导数在 $[a,b]$ 上连续的函数 f:

中点法则

$$\int_{x_0}^{x_1} f(x)\mathrm{d}x = 2hf(w) + \frac{h^3}{3}f''(c). \tag{5.26}$$

这里 $h=(x_1-x_0)/2, w$ 是中点 x_0+h, c 在 x_0 和 x_1 之间.

中点法则对减小需要求值的函数的个数也是有用的. 与梯形法则相比, 相同阶的闭 Newton-Cotes 方法需要求一个函数的值而不是两个, 代价是误差项是原来的 4 倍大.

(5.26) 的证明与梯形法则的推导相同. 设 $h=(x_1-x_0)/2$. $f(x)$ 关于区间中点 $w=x_0+h$ 的一阶 Taylor 展开式是

$$f(x) = f(w) + (x-w)f'(w) + \frac{1}{2}(x-w)^2 f''(c_x),$$

这里的 c_x 与 x 有关, 并且在 x_0 和 x_1 之间. 两边积分得

$$\begin{aligned}\int_{x_0}^{x_1} f(x)\mathrm{d}x &= (x_1-x_0)f(w) + f'(w)\int_{x_0}^{x_1}(x-w)\mathrm{d}x + \frac{1}{2}\int_{x_0}^{x_1} f''(c_x)(x-w)^2\mathrm{d}x \\ &= 2hf(w) + 0 + \frac{f''(c)}{2}\int_{x_0}^{x_1}(x-w)^2\mathrm{d}x \\ &= 2hf(w) + \frac{h^3}{3}f''(c),\end{aligned}$$

这里 $x_0 < c < x_1$. 我们再次用了对被积函数的中值定理, 把二阶导数提到积分号外. 这就完成了 (5.26) 的推导.

复合形式的证明留给读者 (习题 12).

复合中点法则

$$\int_a^b f(x)\mathrm{d}x = 2h\sum_{i=1}^{m} f(w_i) + \frac{(b-a)h^2}{6}f''(c), \tag{5.27}$$

这里 $h=(b-a)/(2m), c$ 在 a 和 b 之间. w_i 是 $[a,b]$ 的 m 个相等子区间的中点.

另一个有用的开 Newton-Cotes 法则是

$$\int_{x_0}^{x_4} f(x)\mathrm{d}x = \frac{4h}{3}\left[2f(x_1) - f(x_2) + 2f(x_3)\right] + \frac{14h^5}{45}f^{(4)}(c), \tag{5.28}$$

这里 $h=(x_4-x_0)/4, x_1=x_0+h, x_2=x_0+2h, x_3=x_0+3h$, 并且 $x_0<c<x_4$. 习题 9 和习题 11 要求你确定它的精度并且把它推广为复合法则.

例 5.10 取 $m=10$ 段区间, 用复合中点法则近似 $\int_0^1 \frac{\sin x}{x}\mathrm{d}x$.

首先, 注意, 若在 $x=0$ 处不经过特殊的处理, 就不能对这个问题直接用闭方法. 可以直接应用中点方法. 中点是 $0.05, 0.15, \cdots, 0.95$, 所以由复合中点法则得

$$\int_0^1 f(x)\mathrm{d}x \approx 2(0.05)\sum_{i=1}^{10} f(m_i) \approx 0.946\,2.$$

精确到 8 位的正确答案是 0.946 083 07. ◀

习题 5.2

1. 取 $m=1,2,4$, 用复合梯形法则来近似以下积分, 并通过与准确值比较来计算误差.

(a) $\int_0^1 x^2\mathrm{d}x$; **(b)** $\int_0^{\frac{\pi}{2}} \cos x\mathrm{d}x$; **(c)** $\int_0^1 \mathrm{e}^x\mathrm{d}x$.

2. 取 $m=1,2,4$, 用复合中点法则来近似习题 1 中的积分, 并且报告误差.

3. 取 $m=1,2,4$, 用复合 Simpson 法则来近似习题 1 中的积分, 并且报告误差.

4. 取 $m=1,2,4$, 用复合 Simpson 法则来近似以下积分, 并且报告误差.

(a) $\int_0^1 x\mathrm{e}^x\mathrm{d}x$; **(b)** $\int_0^1 \frac{\mathrm{d}x}{1+x^2}$; **(c)** $\int_0^{\pi} x\cos x\mathrm{d}x$.

5. 对 $\int_0^1 x^4\mathrm{d}x$ 应用 Simpson 法则近似公式, 并且证明近似误差与 (5.22) 的误差项相匹配.

6. 对 Newton 均差插值多项式进行积分以证明公式 (a) (5.18); (b) (5.19).

7. 对于 $\int_{-1}^1 f(x)\mathrm{d}x$, 求以下近似的精度:

(a) $f(1)+f(-1)$; **(b)** $\frac{2}{3}[f(-1)+f(0)+f(1)]$; **(c)** $f(-1/\sqrt{3})+f(1/\sqrt{3})$.

8. 求中点法则的精度.

9. 求 (5.28) 的精度.

10. 求 c_1, c_2, c_3 使法则

$$\int_0^1 f(x)\mathrm{d}x \approx c_1 f(0) + c_2 f(0.5) + c_3 f(1)$$

的精度大于 1（提示: 代入 $f(x)=1, x, x^2$）, 你能发现得到这种结果的方法吗?

11. 建立带有误差项的法则 (5.28) 的复合形式.

12. 证明复合中点法则 (5.27).

13. 求 4 次 Newton-Cotes 法则 (通常称为**Boole 法则**)

$$\int_{x_0}^{x_4} f(x)\mathrm{d}x \approx \frac{2h}{45}(7y_0+32y_1+12y_2+32y_3+7y_4)$$

的精度.

14. 利用 Boole 法则的误差项正比于 $f^{(6)}(c)$ 的事实, 求准确的误差项, 可以通过以下策略:

对 $\int_0^{4h} x^6 \mathrm{d}x$ 计算 Boole 近似, 求出近似误差, 并且把它用 h 和 $f^{(6)}(c)$ 表示出来.

15. 设 $P_3(x)$ 是三次多项式, 再设 $P_2(x)$ 是在 3 个点 $x=-h, 0, h$ 的插值多项式. 直接证明 $\int_{-h}^{h} P_3(x)\mathrm{d}x = \int_{-h}^{h} P_2(x)\mathrm{d}x$. 这个事实对 Simpson 法则有何启示?

计算机问题 5.2

1. 取 $m=16$ 和 $m=32$ 用复合梯形法则近似以下定积分. 与准确积分进行比较, 并且记录这两种误差.

(a) $\int_0^4 \frac{x\mathrm{d}x}{\sqrt{x^2+9}}$; (b) $\int_0^1 \frac{x^3\mathrm{d}x}{x^2+1}$; (c) $\int_0^1 x\mathrm{e}^x\mathrm{d}x$; (d) $\int_1^3 x^2\ln x\mathrm{d}x$;

(e) $\int_0^{\pi} x^2\sin x\mathrm{d}x$; (f) $\int_2^3 \frac{x^3\mathrm{d}x}{\sqrt{x^4-1}}$; (g) $\int_0^{2\sqrt{3}} \frac{\mathrm{d}x}{\sqrt{x^2+4}}$; (h) $\int_0^1 \frac{x\mathrm{d}x}{\sqrt{x^4+1}}$.

2. 对计算机问题 1 中的积分应用复合 Simpson 法则, 取 $m=16$ 及 $m=32$, 并且记录误差.

3. 取 $m=16$ 及 $m=32$, 应用复合梯形法则近似以下定积分:

(a) $\int_0^1 \mathrm{e}^{x^2}\mathrm{d}x$; (b) $\int_0^{\sqrt{\pi}} \sin x^2\mathrm{d}x$; (c) $\int_0^{\pi} \mathrm{e}^{\cos x}\mathrm{d}x$; (d) $\int_0^1 \ln(x^2+1)\mathrm{d}x$;

(e) $\int_0^1 \frac{x\mathrm{d}x}{2\mathrm{e}^x-\mathrm{e}^{-x}}$; (f) $\int_0^{\pi} \cos\mathrm{e}^x\mathrm{d}x$; (g) $\int_0^1 x^x\mathrm{d}x$; (h) $\int_0^{\frac{\pi}{2}} \ln(\cos x+\sin x)\mathrm{d}x$.

4. 取 $m=16$ 及 $m=32$, 对计算机问题 3 中的积分应用复合 Simpson 法则.

5. 取 $m=16$ 及 $m=32$, 对以下反常积分应用复合中点法则:

(a) $\int_0^{\frac{\pi}{2}} \frac{x}{\sin x}\mathrm{d}x$; (b) $\int_0^{\frac{\pi}{2}} \frac{\mathrm{e}^x-1}{\sin x}\mathrm{d}x$; (c) $\int_0^1 \frac{\arctan x}{x}\mathrm{d}x$.

6. $x=a$ 到 $x=b$ 的 $f(x)$ 所定义的曲线弧长由积分 $\int_a^b \sqrt{1+f'(x)^2}\mathrm{d}x$ 给出. 取 $m=32$ 用复合 Simpson 法则近似以下曲线的长度:

(a) $y=x^3, x\in[0,1]$; (b) $y=\tan x, x\in[0,\frac{\pi}{4}]$; (c) $y=\arctan x, x\in[0,1]$.

7. 对计算机问题 1 中的积分, 对于 $h=b-a, h/2, h/4, \cdots, h/2^8$ 计算复合梯形法则的近似误差并绘图. 例如用 MATLAB的`loglog`命令作一个双对数坐标图. 该图的斜率是什么? 它与理论一致吗?

8. 用复合 Simpson 法则而不是用复合梯形法则执行计算机问题 7.

5.3 Romberg 积分

本节开始讨论计算定积分的有效方法, 它可以通过加入数据而延拓到所要求的精度. Romberg 积分是对复合梯形法则应用外推的结果. 回忆一下, 5.1 节中给定一个与步长 h 有关的法则 $N(h)$ 来近似一个量 M, 如果知道这个法则的阶, 就可以对

这个法则进行外推. 等式 (5.24) 说明复合梯形法则关于 h 是二阶的. 所以可以应用前面所定义的外推来得到至少达到 3 阶的新的法则.

更仔细地检查复合梯形法则 (5.24) 的误差, 对一个无穷次可微函数 f, 可以证明

$$\int_a^b f(x)\mathrm{d}x = \frac{h}{2}\left(y_0 + y_m + 2\sum_{i=1}^{m-1} y_i\right) + c_2h^2 + c_4h^4 + c_6h^6 + \cdots, \tag{5.29}$$

这里 c_i 仅与 f 在 a 和 b 的较高阶导数有关, 而与 h 无关. 例如, $c_2 = (f'(a) - f'(b))h^2/12$. 误差中不出现奇次幂项, 当进行外推时, 就给出了额外的好处. 因为没有奇次幂项, 对由复合梯形法则给出二阶公式进行外推就得到一个 4 阶公式, 对结果是 4 阶的公式进行外推就给出一个 6 阶公式, 如此等等.

外推包括公式在 h 求值一次和在半步长 $\frac{h}{2}$ 上求值一次. 为了达到我们的目的, 定义以下步长序列:

$$h_1 = b-a, \quad h_2 = \frac{1}{2}(b-a), \quad h_3 = \frac{1}{4}(b-a), \cdots h_j = \frac{1}{2^{j-1}}(b-a). \tag{5.30}$$

要近似的量是 $M = \int_a^b f(x)\mathrm{d}x$. 定义近似公式 R_{j1} 是使用 h_j 的复合梯形公式. 于是当需要用外推时, $R_{j+1,j}$ 正好就是步长减半的 R_{j1}. 其次, 注意公式的重叠. 在 R_{j1} 和 $R_{j+1,1}$ 中都需要计算 $f(x)$ 的某些相同的函数值. 例如, 我们有

$$R_{11} = \frac{h_1}{2}(f(a) + f(b)),$$

$$R_{21} = \frac{h_2}{2}\left(f(a) + f(b) + 2f\left(\frac{a+b}{2}\right)\right) = \frac{1}{2}R_{11} + h_2 f\left(\frac{a+b}{2}\right).$$

通过归纳法 (见习题 5) 证明

$$R_{j1} = \frac{1}{2}R_{j-1,1} + h_j\sum_{i=1}^{2^{j-2}} f(a + (2i-1)h_j), \quad j = 2, 3, \cdots. \tag{5.31}$$

等式 (5.31) 给出了提高精度计算复合梯形法则的有效方法. Romberg 积分的第二个特征是外推, 表格格式为:

$$\begin{array}{llll} R_{11} & & & \\ R_{21} & R_{22} & & \\ R_{31} & R_{32} & R_{33} & \\ R_{41} & R_{42} & R_{43} & R_{44} \\ \vdots & & & \ddots \end{array} \tag{5.32}$$

这里我们把第二列 R_{i2} 定义为第一列的外推:

$$R_{22}=\frac{2^2R_{21}-R_{11}}{3},\qquad R_{32}=\frac{2^2R_{31}-R_{21}}{3},\qquad R_{42}=\frac{2^2R_{41}-R_{31}}{3}. \tag{5.33}$$

第 3 列由 M 的 4 阶近似组成, 所以能把它们外推成

$$R_{33}=\frac{4^2R_{32}-R_{22}}{4^2-1},\qquad R_{43}=\frac{4^2R_{42}-R_{32}}{4^2-1},\qquad R_{53}=\frac{4^2R_{52}-R_{42}}{4^2-1}, \tag{5.34}$$

等等, 用公式 (见习题 6) 给出第 j 行第 k 列的元素

$$R_{jk}=\frac{4^{k-1}R_{j,k-1}-R_{j-1,k-1}}{4^{k-1}-1}. \tag{5.35}$$

这个表格是向下和对角无限延伸的下三角矩阵. 对定积分 M 的最好近似是 R_{jj}, 即至今算得的底部最右边的值, 它是 $2j$ 阶近似. Remberg 积分的计算恰好是循环地写出公式 (5.31) 和公式 (5.32) 的问题.

Romberg 积分

$$R_{11}=(b-a)\frac{f(a)+f(b)}{2}$$

for $j=2,3,\cdots$

$$h_j=\frac{b-a}{2^{j-1}}$$

$$R_{j1}=\frac{1}{2}R_{j-1,1}+h_j\sum_{i=1}^{2^{j-2}}f(a+(2i-1)h_j)$$

for $k=2,\cdots,j,$

$$R_{jk}=\frac{4^{k-1}R_{j,k-1}-R_{j-1,k-1}}{4^{k-1}-1}$$

end

end

MATLAB代码是直截了当地执行前面的算法.

```
%Program 5.1 Romberg integration
% Computes approximation to definite integral
% Inputs: Matlab inline function specifying integrand f,
%  a,b integration interval, n=number of rows
% Output: Romberg tableau r
function r=romberg(f,a,b,n)
h=(b-a)./(2.^(0:n-1));
r(1,1)=(b-a)*(f(a)+f(b))/2;
for j=2:n
  subtotal = 0;
```

```
      for i=1:2^(j-2)
        subtotal = subtotal + f(a+(2*i-1)*h(j));
      end
      r(j,1) = r(j-1,1)/2+h(j)*subtotal;
      for k=2:j
        r(j,k)=(4^(k-1)*r(j,k-1)-r(j-1,k-1))/(4^(k-1)-1);
      end
    end
```

例 5.11 用 Romberg 积分近似 $\int_1^2 \ln x\,\mathrm{d}x$.

运行上面的代码, 结果是

```
>> romberg(inline('log(x)'),1,2,4)

ans =

   0.34657359027997   0                  0                  0
   0.37601934919407   0.38583460216543   0                  0
   0.38369950940944   0.38625956281457   0.38628789352451 0
   0.38564390995210   0.38629204346631   0.38629420884310 0.38629430908625
```

注意 R_{43} 和 R_{44} 的前面 6 位小数是一致的. 这是 Romberg 方法收敛于定积分的正确值的一个信号. 可以把它们与精确值 $2\ln 2 - 1 \approx 0.386\,294\,36$ 比较. ◀

把例 5.11 的结果与例 5.8 的结果进行比较, 表明 Romberg 积分的第二列的最后一个值与复合 Simpson 法则的结果相匹配. 这并非巧合. 事实上, 恰如 Romberg 积分的第一列是由逐次的复合梯形法则的值所定义, 第二列是复合 Simpson 法则的值. 换言之, 复合梯形法则的外推是复合 Simpson 法则 (见习题 3).

Romberg 积分通常的停止准则是计算新的行, 直到对角线上的两个连续的值 R_{jj} 小于预设的误差容限.

习题 5.3

1. 对以下积分用 Romberg 积分求 R_{33}:

   (a) $\int_0^1 x^2\mathrm{d}x$; (b) $\int_0^{\frac{\pi}{2}} \cos x\mathrm{d}x$; (c) $\int_0^1 \mathrm{e}^x\mathrm{d}x$.

2. 对以下积分用 Romberg 积分求 R_{33}:

   (a) $\int_0^1 x\mathrm{e}^x\mathrm{d}x$; (b) $\int_0^1 \frac{\mathrm{d}x}{1+x^2}$; (c) $\int_0^\pi x\cos x\mathrm{d}x$.

3. 证明对 R_{11} 和 R_{21} 中的复合梯形法则进行外推产生 R_{22} 中的复合 Simpson 法则 (步长为 h_2).

4. 证明 Romberg 积分的 R_{33} 可以表示为习题 5.2.13 中定义的 Boole 法则 (步长为 h_3).

5. 证明公式 (5.31).

6. 证明公式 (5.35).

计算机问题 5.3

1. 用 Romberg 积分近似 R_{55} 来近似以下定积分. 与准确的积分进行比较, 记录误差.

(a) $\displaystyle\int_0^4 \frac{x\mathrm{d}x}{\sqrt{x^2+9}}$; (b) $\displaystyle\int_0^1 \frac{x^3\mathrm{d}x}{x^2+1}$; (c) $\displaystyle\int_0^1 x\mathrm{e}^x\mathrm{d}x$; (d) $\displaystyle\int_1^3 x^2\ln x\mathrm{d}x$;

(e) $\displaystyle\int_0^\pi x^2\sin x\mathrm{d}x$; (f) $\displaystyle\int_2^3 \frac{x^3\mathrm{d}x}{\sqrt{x^4-1}}$; (g) $\displaystyle\int_0^{2\sqrt{3}} \frac{\mathrm{d}x}{\sqrt{x^2+4}}$; (h) $\displaystyle\int_0^1 \frac{x\mathrm{d}x}{\sqrt{x^4+1}}$.

2. 用 Romberg 积分来近似以下定积分. 作为停止准则, 继续到对角线上的两个连续的值小于 0.5×10^{-8} 为止.

(a) $\displaystyle\int_0^1 \mathrm{e}^{x^2}\mathrm{d}x$; (b) $\displaystyle\int_0^{\sqrt{\pi}} \sin x^2\mathrm{d}x$; (c) $\displaystyle\int_0^\pi \mathrm{e}^{\cos x}\mathrm{d}x$; (d) $\displaystyle\int_0^1 \ln(x^2+1)\mathrm{d}x$;

(e) $\displaystyle\int_0^1 \frac{x\mathrm{d}x}{2\mathrm{e}^x-\mathrm{e}^{-x}}$; (f) $\displaystyle\int_0^\pi \cos\mathrm{e}^x\mathrm{d}x$; (g) $\displaystyle\int_0^1 x^x\mathrm{d}x$; (h) $\displaystyle\int_0^{\frac{\pi}{2}} \ln(\cos x+\sin x)\mathrm{d}x$.

3. (a) 测试 Romberg 积分第二列的阶. 如果它们是 4 阶近似, 误差的双对数坐标与 h 的关系图看起来如何? 对例 5.11 中的积分, 执行这一过程.

   (b) 测试 Romberg 积分第 3 列的阶.

5.4 自适应求积

到目前为止, 我们学习的近似积分方法用相等的步长. 一般地, 较小的步长可以提高精度. 变化很大的函数需要更多的步数, 因此需要更多的计算时间, 这是因为需要较小的步长来追踪函数的变化.

尽管对复合方法我们有误差公式, 但是用它们直接计算满足所给定的误差容限的 h 值常常是困难的. 这些公式包含可能复杂且难以在问题的整个区间估计的较高导数. 如果只是通过函数值的表格了解这个函数, 那么其较高阶导数甚至可能得不到.

应用等步长的复合公式的第二个问题是, 函数在其定义域的某些部分变化剧烈而在其他部分变化平缓 (见图 5-5). 在前半部分中一种步长足以满足误差容限, 但在后半部分可能全都无效.

幸运的是, 有方法解决这两个问题. 利用来自积分误差公式的信息, 我们可以建立一种准则来确定在计算中什么样的步长适合于特定的子区间. 隐藏在这种称为自适应求积 (Adaptive Quadrature) 的方法后面的思想与我们在本章学习过的外推思想密切相关.

根据 (5.21), 在区间 $[a,b]$ 上的梯形法则 $S_{[a,b]}$ 满足公式

$$\int_a^b f(x)\mathrm{d}x = S_{[a,b]} - h^3\frac{f''(c_0)}{12}, \tag{5.36}$$

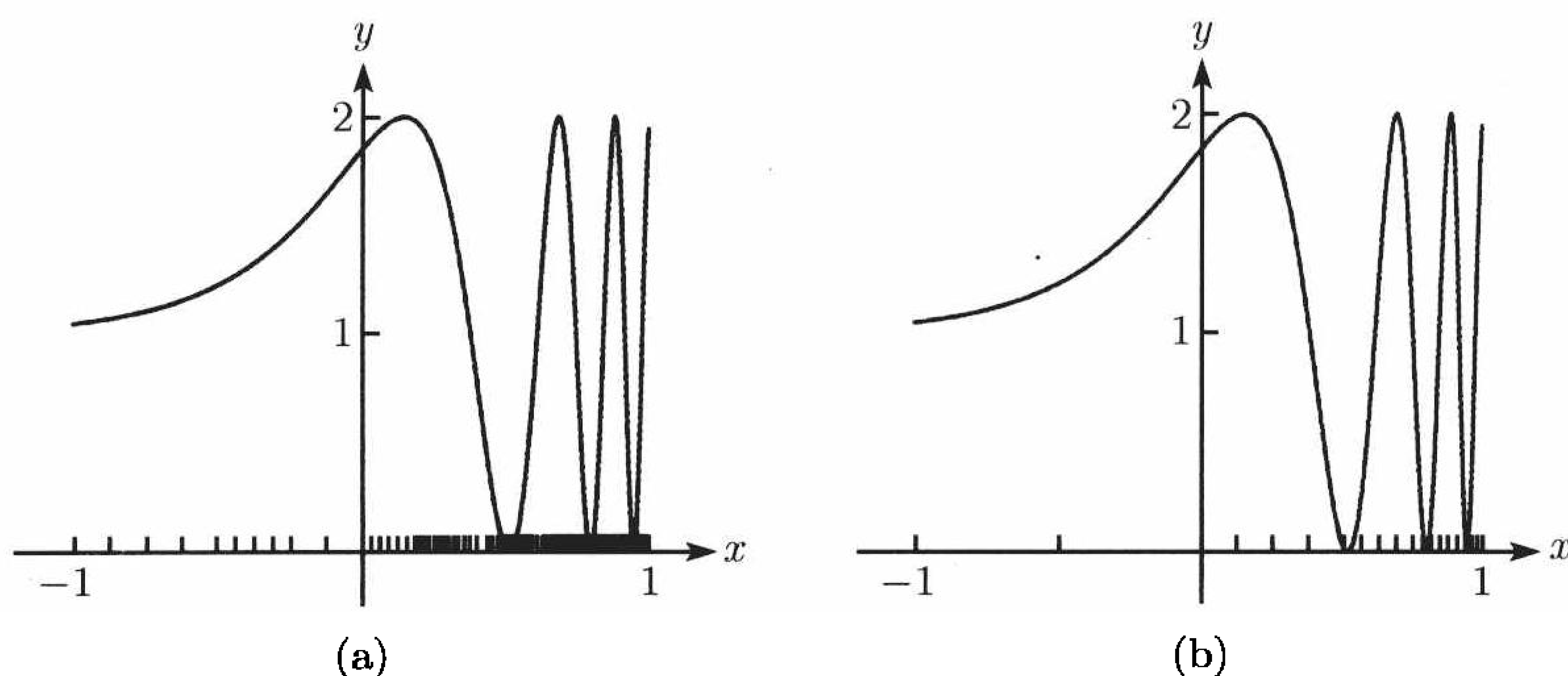


图 5-5　用于 $f(x)=1+\sin \mathrm{e}^{3x}$ 的自适应求积. 误差容限 TOL = 0.005. (a) 自适应梯形法则需要 140 个子区间; (b) 自适应 Simpson 法则需要 20 个子区间

它对某个 $a<c_0<b$ 成立, 这里 $h=b-a$. 设 c 是 $[a,b]$ 的中点, 我们可以对两个半区间用梯形法则, 根据同样的公式得到

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= S_{[a,c]}-\frac{h^3}{8}\frac{f''(c_1)}{12}+S_{[c,b]}-\frac{h^3}{8}\frac{f''(c_2)}{12}\\ &= S_{[a,c]}+S_{[c,b]}-\frac{h^3}{4}\frac{f''(c_3)}{12},\end{aligned} \tag{5.37}$$

这里 c_1 及 c_2 分别在 $[a,c]$ 及 $[c,b]$ 内. 我们已经用定理 5.1 合并误差项. 从 (5.36) 减去 (5.37) 得到

$$S_{[a,b]}-(S_{[a,c]}+S_{[c,b]})=-\frac{h^3}{4}\frac{f''(c_3)}{12}+h^3\frac{f''(c_0)}{12}\approx\frac{3}{4}h^3\frac{f''(c_3)}{12}, \tag{5.38}$$

这里已经设 $f''(c_3)\approx f''(c_0)$.

从等式减去精确的积分, 我们已经把误差 (近似地) 写为用我们能够计算的东西来表示. 例如, 注意到, 由 (5.37) 式 $S_{[a,b]}-(S_{[a,c]}+S_{[c,b]})$ 在区间 $[a,b]$ 上近似地是 $S_{[a,c]}+S_{[c,b]}$ 积分误差的 3 倍. 因此, 可以检查前一个表达式是否小于`3*TOL` (即为某个误差容限), 来作为检查后一个近似未知的精确积分是否在 TOL 之内的一种方法.

如果不符合这个准则, 我们可以再细分. 既然有了在给定的子区间上认可近似的准则, 我们就可以继续把区间分半, 并且对这种半区间循环地应用这种准则. 每分半一次, 所需要的误差容限由因子 2 减小, 而误差 (对于梯形法则) 应该由因子 $2^3=8$ 减小, 所以可以使用自适应方法通过足够的区间分半次数来满足最初的误差容限.

自适应求积

在误差容限 TOL 之内近似求 $\int_a^b f(x)\mathrm{d}x$:

$$c=\frac{a+b}{2},$$

$$S_{[a,b]}=(b-a)\frac{f(a)+f(b)}{2},$$

if $|S_{[a,b]}-S_{[a,c]}-S_{[c,b]}|<3\cdot\text{TOL}\left(\frac{b-a}{b_{\text{orig}}-a_{\text{orig}}}\right).$

接受 $S_{[a,c]}+S_{[c,b]}$ 作为整个区间 $[a,b]$ 上的近似.

else

对 $[a,c]$ 及 $[c,b]$ 循环地重复上面步骤.

end

MATLAB编程策略操作如下. 建立一张逐渐形成的子区间列表. 这张表最初由一个区间 $[a,b]$ 组成. 通常选取表中最后一个子区间并使用这个准则. 如果满足, 把这一子区间上的积分近似式加入运行求和, 并且在表中删除这个区间. 如果不满足, 则在表中由两个子区间来代替这个子区间, 列表长度加一, 进行到列表的最后并重复上面的步骤. 下面的代码执行了这种策略.

```
%Program 5.2 Adaptive Quadrature
% Computes approximation to definite integral
% Inputs: Matlab inline function f, interval [a0,b0],
%  error tolerance tol0
% Output: approximate definite integral
function sum=adapquad(f,a0,b0,tol0)
sum=0; n=1; a(1)=a0; b(1)=b0; tol(1)=tol0; app(1)=trap(f,a,b);
while n>0               % n is current position at end of the list
    c=(a(n)+b(n))/2; oldapp=app(n);
    app(n)=trap(f,a(n),c);app(n+1)=trap(f,c,b(n));
    if abs(oldapp-(app(n)+app(n+1)))<3*tol(n)
        sum=sum+app(n)+app(n+1);       % success
        n=n-1;                         % done with interval
    else                               % divide into two intervals
        b(n+1)=b(n); b(n)=c;           % set up new intervals
        a(n+1)=c;
        tol(n)=tol(n)/2; tol(n+1)=tol(n);
        n=n+1;                         % go to end of list, repeat
    end
end

function s=trap(f,a,b)
s=(f(a)+f(b))*(b-a)/2;
```

例 5.12 用自适应求积法近似计算积分

$$\int_{-1}^{1}(1+\sin e^{3x})dx.$$

图 5-5a 表示在误差容限 0.005 内对 $f(x)$ 实施自适应求积算法的结果. 尽管需

要 140 个子区间, 但是它们中间仅有 11 个落在 "平稳" 区域 $[-1,0]$. 近似定积分是 2.502 ± 0.005. 在第二轮中, 我们把误差容限改为 0.5×10^{-4} 并得到 2.500 8, 这是在超过 1 316 个子区间上进行的计算, 有 4 位小数可信. ◀

当然, 可以用更复杂的法则来代替梯形法则. 例如, 用 $S_{[a,b]}$ 代表在区间 $[a,b]$ 上的 Simpson 法则 (5.22):

$$\int_a^b f(x)\mathrm{d}x = S_{[a,b]} - \frac{h^5}{90}f^{(4)}(c_0). \tag{5.39}$$

在 $[a,b]$ 的两个半子区间内应用 Simpson 法则得到

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= S_{[a,b]} - \frac{h^5}{32}\frac{f^{(4)}(c_1)}{90} + S_{[c,b]} - \frac{h^5}{32}\frac{f^{(4)}(c_2)}{90} \\ &= S_{[a,c]} + S_{[c,b]} - \frac{h^5}{16}\frac{f^{(4)}(c_3)}{90},\end{aligned} \tag{5.40}$$

这里用了定理 5.1 把两个误差项结合在一起. 用 (5.39) 减去 (5.40), 得到

$$S_{[a,b]} - (S_{[a,c]} + S_{[c,b]}) = h^5\frac{f^{(4)}(c_0)}{90} - \frac{h^5}{16}\frac{f^{(4)}(c_3)}{90} \approx \frac{15}{16}h^5\frac{f^{(4)}(c_3)}{90}, \tag{5.41}$$

这里假设 $f^{(4)}(c_3) \approx f^{(4)}(c_0)$.

因为 $S_{[a,b]} - (S_{[a,c]} + S_{[c,b]})$ 是近似 $S_{[a,c]} + S_{[c,b]}$ 的误差的 15 倍, 所以可以得到新的准则

$$|S_{[a,b]} - (S_{[a,c]} + S_{[c,b]})| < 15 * \mathrm{TOL} \tag{5.42}$$

并且如前进行. 按惯例, 在这个准则中用 10 代替 15, 使算法更保守. 图 5-5b 显示了对同一个积分应用自适应 Simpson 求积方法的结果. 当使用误差容限 0.005 时, 近似积分是 2.500, 用了 20 个子区间, 比自适应梯形法则求积法节省了很多. 把误差容限减小到 0.5×10^{-4}, 就得到 2.500 8, 只用了 58 个子区间.

习题 5.4

1. 取误差容限 TOL=0.05, 利用梯形法则, 手工用自适应求积方法, 求下列积分的近似值及近似误差.

   (a) $\int_0^1 x^2\mathrm{d}x$; (b) $\int_0^{\frac{\pi}{2}} \cos x\mathrm{d}x$; (c) $\int_0^1 \mathrm{e}^x\mathrm{d}x$.

2. 取误差容限 TOL=0.01, 利用 Simpson 法则, 手工用自适应求积方法, 求下列积分的近似值及近似误差.

   (a) $\int_0^1 x\mathrm{e}^x\mathrm{d}x$; (b) $\int \frac{\mathrm{d}x}{1+x^2}$; (c) $\int_0^{\pi} x\cos x\mathrm{d}x$.

3. 对中点法则 (5.26) 建立自适应求积方法. 从求子区间上一条满足误差容限的准则开始.
4. 对法则 (5.28) 建立自适应求积方法.

计算机问题 5.4

1. 用自适应梯形求积方法, 误差在 0.5×10^{-8} 之内, 求以下定积分的近似值, 用 8 位准确小数记录答案, 并记录所需要的子区间个数.

   (a) $\int_0^4 \frac{x\mathrm{d}x}{\sqrt{x^2+9}}$; (b) $\int_0^1 \frac{x^3\mathrm{d}x}{x^2+1}$; (c) $\int_0^1 x\mathrm{e}^x\mathrm{d}x$; (d) $\int_1^3 x^2\ln x\mathrm{d}x$;

   (e) $\int_0^\pi x^2\sin x\mathrm{d}x$; (f) $\int_2^3 \frac{x^3\mathrm{d}x}{\sqrt{x^4-1}}$; (g) $\int_0^{2\sqrt{3}} \frac{\mathrm{d}x}{\sqrt{x^2+4}}$; (h) $\int_0^1 \frac{x\mathrm{d}x}{\sqrt{x^4+1}}$.
2. 把自适应梯形法则求积方法的 MATLAB代码修改成使用 Simpson 法则的自适应求积方法. 采用以 10 代替 15 的准则 (5.42). 求例 5.12 中的积分的近似值, 误差在 0.005 之内, 并且与图 5-5b 进行比较, 需要多少个子区间?
3. 对计算机问题 2 中建立的自适应 Simpson 法则, 执行计算机问题 1 中的步骤.
4. 用在习题 3 中建立的自适应中点法则, 执行计算机问题 1 中的步骤.
5. 用在习题 4 中建立的自适应开 Newton-Cotes 法则, 执行计算机问题 1 中的步骤.
6. 用自适应梯形求积法, 计算以下定积分的近似值, 误差在 0.5×10^{-8} 之内.

   (a) $\int_0^1 \mathrm{e}^{x^2}\mathrm{d}x$; (b) $\int_0^{\sqrt{\pi}} \sin x^2\mathrm{d}x$; (c) $\int_0^\pi \mathrm{e}^{\cos x}\mathrm{d}x$; (d) $\int_0^1 \ln(x^2+1)\mathrm{d}x$;

   (e) $\int_0^1 \frac{x\mathrm{d}x}{2\mathrm{e}^x-\mathrm{e}^{-x}}$; (f) $\int_0^\pi \cos\mathrm{e}^x\mathrm{d}x$; (g) $\int_0^1 x^x\mathrm{d}x$; (h) $\int_0^{\frac{\pi}{2}} \ln(\cos x+\sin x)\mathrm{d}x$.
7. 用自适应 Simpson 求积方法, 执行计算机问题 6 中的步骤.
8. 在正态分布平均的标准差 σ 之内概率是 $\frac{1}{\sqrt{2\pi}}\int_{-\sigma}^{\sigma}\mathrm{e}^{-x^2/2}\mathrm{d}x$. 用自适应 Simpson 求积方法, 求准确到 8 位小数的概率, 在以下标准差内:

   (a) 1; (b) 2; (c) 3.
9. 写出称为`myerf.m`的 MATLAB函数, 它对任意输入 x, 用自适应 Simpson 法则准确到 8 位小数计算下面的值: $\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x \mathrm{e}^{-s^2}\mathrm{d}s$. 通过与 MATLAB函数`erf`进行比较, 对 $x=1$ 及 $x=3$ 试验你的程序.

5.5 Gauss 求积

求积方法的精度是用这种方法求积没有误差的所有多项式的次数. n 次的 Newton-Cotes 方法有精度 n（n 是奇数）及 $n+1$（n 是偶数）, 梯形法则 ($n=1$ 的 Newton-Cotes 方法) 有精度 1. Simpson 法则 ($n=2$) 对次数小于等于 3 的多项式是正确的.

为了达到这种精度, Newton-Cotes 公式在等距点用 $n+1$ 个函数求值. 我们要

求的问题使我们想起第 3 章关于 Chebyshev 多项式的讨论. Newton-Cotes 公式就其精度而言是否最优? 或者能否建立更有功效的公式? 特别是, 如果放弃等距点上的求值, 还有更好的方法吗?

至少从精度的观点来看, 还有更有功效和复杂的方法. 本节挑选出其中最著名的一种方法来讨论. 当用 $n+1$ 个点时, Gauss 求积方法有精度 $2n+1$, 是 Newton-Cotes 方法的 2 倍. 它的求值点不是等距的. 解释 Gauss 求积方法如何工作要涉及正交函数, 这不仅因为它本身有趣, 还因为它是数值方法受益于正交性启发和鼓舞的冰山一角.

定义 5.3 区间 $[a,b]$ 上的一组非零函数 $\{p_0,p_1,\cdots,p_n\}$ 如果满足

$$\int_a^b p_j(x)p_k(x)\mathrm{d}x = \begin{cases} 0, & j \neq k, \\ \neq 0, & j = k, \end{cases}$$

则它们在区间 $[a,b]$ 上是**正交的**(orthogonal).

定理 5.4 如果 $\{p_0,p_1,\cdots,p_n\}$ 是区间 $[a,b]$ 上的一组正交多项式, 这里 p_i 的次数是 i, 那么 $\{p_0,p_1,\cdots,p_n\}$ 是 $[a,b]$ 上次数最高为 n 的多项式向量空间的一组基.

证 我们必须证明这些多项式张成该向量空间而且是线性无关的. 容易的归纳论证表明, 任一组多项式 $\{p_0,p_1,\cdots,p_n\}$ (这里 p_i 次数是 i)张成了次数最多是 n 的多项式空间. 为了证明线性无关, 假设存在线性相关性 $\sum\limits_{i=0}^{n} c_i p_i(x) = 0$, 并且用正交性假设证明所有的 c_i 必须是零. 对任意 $0 \leqslant k \leqslant n$, 因为 p_k 与除自己以外的每一个多项式正交, 我们得到

$$0 = \int_a^b p_k \sum_{i=0}^{n} c_i p_i(x)\mathrm{d}x = \sum_{i=0}^{n} c_i \int_a^b p_k p_i \mathrm{d}x = c_k \int_a^b p_k^2 \mathrm{d}x. \tag{5.43}$$

所以 $c_k = 0$.

亮点 正交性

在第 4 章, 我们发现有限维向量的正交性有助于阐述和求解最小二乘问题. 对于求积, 像单变量多项式的向量空间那样, 我们需要无穷维空间中的正交性. 一组基是单项式基 $\{1,x,x^2,\cdots,\}$. 然而, 更有用的基是一个正交集. 对于在区间 $[-1,1]$ 的正交性, 其正确的选择是 Legendre 多项式.

我们省略以下定理的证明:

定理 5.5 如果 $\{p_0,\cdots,p_n\}$ 是 $[a,b]$ 上的多项式的一个正交集, 并且如果 p_i 的次数是 i, 那么 p_i 在区间 (a,b) 中有 i 个不同的根.

例 5.13 在区间 $[-1,1]$ 上找 3 个正交多项式.

最佳初始估计是 $p_0(x)=1$ 及 $p_1(x)=x$, 这是因为

$$\int_{-1}^{1} 1\cdot x\mathrm{d}x=0.$$

试验 $p_2(x)=x^2$ 不太成功, 因为它与 $p_0(x)$ 缺乏正交性:

$$\int_{-1}^{1} p_0(x)x^2\mathrm{d}x=\frac{2}{3}\neq 0.$$

修改 $p_2(x)=x^2+c$, 我们发现只要 $c=-\frac{1}{3}$,

$$\int_{-1}^{1} p_0(x)(x^2+c)\mathrm{d}x=\frac{2}{3}+2c=0.$$

经检验, p_1 与 p_2 是正交的 (见习题 7), 因此集合 $\{1,x,x^2-\frac{1}{3}\}$ 是在 $[-1,1]$ 上的一组正交集. ◀

例 5.13 中的 3 个多项式是 Legendre 发现的一组正交多项式.

例 5.14 证明Legendre 多项式集合

$$p_i(x)=\frac{1}{2^i i!}\frac{\mathrm{d}^i}{\mathrm{d}x^i}[(x^2-1)^i],\quad 0\leqslant i\leqslant n$$

在 $[-1,1]$ 上正交.

首先注意, $p_i(x)$ 是 i 次多项式 (作为 $2i$ 次多项式的 i 阶导数). 其次注意, 如果 $i<j$, 则 $(x^2-1)^j$ 的 i 阶导数可被 (x^2-1) 整除.

我们要证明, 如果 $i<j$, 那么积分

$$\int_{-1}^{1}[(x^2-1)^i]^{(i)}[(x^2-1)^j]^{(j)}\mathrm{d}x$$

等于 0. 取 $u=[(x^2-1)^i]^{(i)}$, $\mathrm{d}v=[(x^2-1)^j]^{(j)}\mathrm{d}x$ 进行分部积分, 得到

$$uv-\int_{-1}^{1}v\mathrm{d}u=[(x^2-1)^i]^{(i)}[(x^2-1)^j]^{(j-1)}\Big|_{-1}^{1}-\int_{-1}^{1}[(x^2-1)^i]^{(i+1)}[(x^2-1)^j]^{(j-1)}\mathrm{d}x$$

$$=-\int_{-1}^{1}[(x^2-1)^i]^{(i+1)}[(x^2-1)^j]^{(j-1)}\mathrm{d}x,$$

这是因为 $[(x^2-1)^j]^{(j-1)}$ 被 (x^2-1) 整除.

重复 $i+1$ 次分部积分后, 留下

$$(-1)^{i+1}\int_{-1}^{1}[(x^2-1)^i]^{(2i+1)}[(x^2-1)^j]^{(j-i-1)}\mathrm{d}x=0,$$

这是因为 $(x^2-1)^i$ 的 $(2i+1)$ 阶导数是零. ◀

根据定理 5.5, n 次 Legendre 多项式在 $[-1,1]$ 中有 n 个根 $x_1,\cdots,x_n$. 函数的 Gauss 求积法是简单地把函数在 Legendre 根处求值后进行线性组合. 我们通过用结点是在 Legendre 根处的插值多项式的积分来近似所求函数的积分, 以此达到这个目的.

固定 n, 并且设 $Q(x)$ 是被积函数 $f(x)$ 在结点 $x_1,\cdots,x_n$ 的插值多项式. 利用 Lagrange 公式, 可将 $Q(x)$ 写为

$$Q(x)=\sum_{i=1}^{n}L_i(x)f(x_i),\quad 其中\quad L_i(x)=\frac{(x-x_1)\cdots\overline{(x-x_i)}\cdots(x-x_n)}{(x_i-x_1)\cdots\overline{(x_i-x_i)}\cdots(x_i-x_n)}.$$

两边积分得到积分的以下近似:

Gauss 求积

$$\int_{-1}^{1}f(x)\mathrm{d}x\approx\sum_{i=1}^{n}c_if(x_i),\tag{5.44}$$

这里

$$c_i=\int_{-1}^{1}L_i(x)\mathrm{d}x,\qquad i=1,\cdots,n.$$

c_i 以极高的精度被列出. 表 5-2 给出了直到 $n=4$ 的值.

表 5-2 Gauss 求积系数: n 次 Legendre 多项式的根 x_i 以及 (5.44) 中的系数 c_i

n	根 x_i	系数 c_i
2	$-\sqrt{1/3}=-0.577\,350\,269\,189\,63$	$1=1.000\,000\,000\,000\,00$
	$\sqrt{1/3}=0.577\,350\,269\,189\,63$	$1=1.000\,000\,000\,000\,00$
3	$-\sqrt{3/5}=-0.774\,596\,669\,241\,48$	$5/9=0.555\,555\,555\,555\,55$
	$0=0.000\,000\,000\,000\,00$	$8/9=0.888\,888\,888\,888\,88$
	$\sqrt{3/5}=0.774\,596\,669\,241\,48$	$5/9=0.555\,555\,555\,555\,55$
4	$-\sqrt{\frac{15+2\sqrt{30}}{35}}=-0.861\,136\,311\,594\,05$	$\frac{90-5\sqrt{30}}{180}=0.347\,854\,845\,137\,45$
	$-\sqrt{\frac{15-2\sqrt{30}}{35}}=-0.339\,981\,043\,584\,86$	$\frac{90+5\sqrt{30}}{180}=0.652\,145\,154\,862\,55$
	$\sqrt{\frac{15-2\sqrt{30}}{35}}=0.339\,981\,043\,584\,86$	$\frac{90+5\sqrt{30}}{180}=0.652\,145\,154\,862\,55$
	$\sqrt{\frac{15+2\sqrt{30}}{35}}=0.861\,136\,311\,594\,05$	$\frac{90-5\sqrt{30}}{180}=0.347\,854\,845\,137\,45$

例 5.15 用 Gauss 求积法近似

$$\int_{-1}^{1}\mathrm{e}^{-\frac{x^2}{2}}\mathrm{d}x.$$

14 位数字的准确答案是 1.711 248 783 784 30. 对于被积函数 $f(x)=\mathrm{e}^{-\frac{x^2}{2}}$, 那么 $n=2$ 的 Gauss 求积近似是

$$\int_{-1}^{1}\mathrm{e}^{-\frac{x^2}{2}}\mathrm{d}x\approx c_1f(x_1)+c_2f(x_2)=1\cdot f\left(-\sqrt{\frac{1}{3}}\right)+1\cdot f\left(\sqrt{\frac{1}{3}}\right)\approx 1.692\,963\,449\,781\,23.$$

$n=3$ 的近似是

$$\frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right)+\frac{8}{9}f(0)+\frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right)\approx 1.712\,020\,245\,201\,91.$$

$n=4$ 的近似是

$$c_1f(x_1)+c_2f(x_2)+c_3f(x_3)+c_4f(x_4)\approx 1.711\,224\,504\,599\,49.$$

这个近似 (用了 4 个函数的求值) 比 Romberg 近似 R_{33}(它用了在 $[-1,1]$ 上 5 个等距的函数求值) 要接近于准确值.

```
1.21306131942527   0                  0
1.60653065971263   1.73768710647509   0
1.68576223244091   1.71217275668367   1.71047180003091
```

Gauss 求积法精确性的秘密由下面的定理来揭示.

定理 5.6 利用 $[-1,1]$ 上的 n 次 Legendre 多项式的 Gauss 求积方法有精度 $2n-1$.

证 设 $P(x)$ 是次数最多为 $2n-1$ 的多项式. 我们必须证明它用 Gauss 求积法可以准确地求得积分.

利用多项式的长除法, 可将 $P(x)$ 表示为

$$P(x)=S(x)p_n(x)+R(x),\tag{5.45}$$

这里 $S(x)$ 和 $R(x)$ 是次数小于 n 的多项式. 注意, Gauss 求积对多项式 $R(x)$ 将是准确的, 这是因为它恰好是 $n-1$ 次插值多项式的积分, 而这个插值多项式恒等于 $R(x)$.

在 n 次 Legendre 多项式的根 x_i 处, 因为对所有的 i 有 $p_n(x_i)=0$, 所以 $P(x_i)=R(x_i)$, 这意味着 Gauss 求积近似将是相同的. 但是它们的积分也是相等的: 积分 (5.45) 得

$$\int_{-1}^{1}P(x)\mathrm{d}x=\int_{-1}^{1}S(x)p_n(x)\mathrm{d}x+\int_{-1}^{1}R(x)\mathrm{d}x=0+\int_{-1}^{1}R(x)\mathrm{d}x,$$

这是因为根据定理 5.4, $S(x)$ 可以写成低于 n 次的多项式的线性组合, 它与 $p_n(x)$ 是正交的. 因为 Gauss 求积对于 $R(x)$ 是准确的, 所以它对 $P(x)$ 也必定是准确的.

要近似计算在一般的区间 $[a,b]$ 上的积分, 问题需要转化到 $[-1,1]$ 上去. 利用代换 $t=(2x-a-b)/(b-a)$, 容易验证

$$\int_a^b f(x)\mathrm{d}x=\int_{-1}^1 f\left(\frac{(b-a)t+b+a}{2}\right)\frac{b-a}{2}\mathrm{d}t. \tag{5.46}$$

我们用一个例子来说明.

例 5.16 用 Gauss 求积法, 求 $\int_1^2 \ln x\mathrm{d}x$ 的近似值.

根据 (5.46),

$$\int_1^2 \ln x\ \mathrm{d}x=\int_{-1}^1 \ln\left(\frac{t+3}{2}\right)\frac{1}{2}\mathrm{d}t.$$

现在可设 $f(t)=\ln((t+3)/2)/2$, 并用标准的根与系数. $n=4$ 的结果是 0.386 294 496 938 71, 与准确值 $2\ln 2-1\approx 0.386\ 294\ 361\ 119\ 89$ 进行比较. 它又一次比例 5.11 中用 4 个点的 Romberg 积分更精确. ◀

习题 5.5

1. 用 $n=2$ 的 Gauss 求积法近似以下积分. 与准确值比较, 并且给出近似误差.

(a) $\int_{-1}^1(x^3+2x)\mathrm{d}x$; (b) $\int_{-1}^1 x^4\mathrm{d}x$; (c) $\int_{-1}^1 \mathrm{e}^x\mathrm{d}x$; (d) $\int_{-1}^1 \cos\pi x\mathrm{d}x$.

2. 用 $n=3$ 的 Gauss 求积法近似习题 1 中的积分, 并给出误差.

3. 用 $n=4$ 的 Gauss 求积法近似习题 1 中的积分, 并给出误差.

4. 利用代换 (5.46), 改变变量, 重新写成 $[-1,1]$ 上的积分.

(a) $\int_0^4 \frac{x\mathrm{d}x}{\sqrt{x^2+9}}$; (b) $\int_0^1 \frac{x^3\mathrm{d}x}{x^2+1}$; (c) $\int_0^1 x\mathrm{e}^x\mathrm{d}x$; (d) $\int_1^3 x^2\ln x\mathrm{d}x$.

5. 用 $n=4$ 的 Gauss 求积法, 近似习题 4 中的积分.

6. 用 $n=4$ 的 Gauss 求积法近似以下积分.

(a) $\int_0^1(x^3+2x)\mathrm{d}x$; (b) $\int_1^4 \ln x\mathrm{d}x$; (c) $\int_{-1}^2 x^5\mathrm{d}x$; (d) $\int_{-3}^3 \mathrm{e}^{-\frac{x^2}{2}}\mathrm{d}x$.

7. 证明 Legendre 多项式 $p_1(x)=x$ 与 $p_2(x)=x^2-\frac{1}{3}$ 在 $[-1,1]$ 上正交.

8. 求直到 3 次的 Legendre 多项式并与例 5.13 比较.

9. 对 $n=3$, 验证表 5-2 中的系数 c_i 及 x_i.

10. 对 $n=4$, 验证表 5-2 中的系数 c_i 及 x_i.

实例检验 5 计算机辅助建模中的运动控制

计算机辅助建模和制造需要对沿着指定的运动路径的空间位置进行精确的控制. 我们将说明用自适应求积法解决这个问题的基本部分：等分或者把任意路径分成等长的子路径.

在数值机器问题中, 最好是沿着路径保持恒定速度. 在每一秒中, 行程应沿着等长的机器–物质界面进行. 在其他的运动计划应用中, 包括计算机动画, 可能需要更复杂的前进曲线：用

手去抓门的把手时可能开始和结束的速度很慢, 但是在中间的速度较快. 机器人学与虚拟现实应用需要构造要通过的参数化曲线与曲面. 构造小幅度的等增量路径的表格常常是必要的第一步.

假设给定参数路径 $P=\{x(t),y(t)|0\leqslant t\leqslant 1\}$. 图 5-6 显示了示例的路径

$$P=\begin{cases} x(t)=0.5+0.3t+3.9t^2-4.7t^3, \\ y(t)=1.5+0.3t+0.9t^2-2.7t^3, \end{cases}$$

这是由 4 个点 $(0.5,1.5),(0.6,1.6),(2,2),(0,0)$ 所确定的 Bézier 曲线. (见 3.5 节) 由相等间隔的参数值 $t=0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1$ 所确定的点如图 5-6 所示. 注意参数的相等间隔并不意味着弧长的相等间隔. 你的目标是用求积方法把这条路径 n 等分.

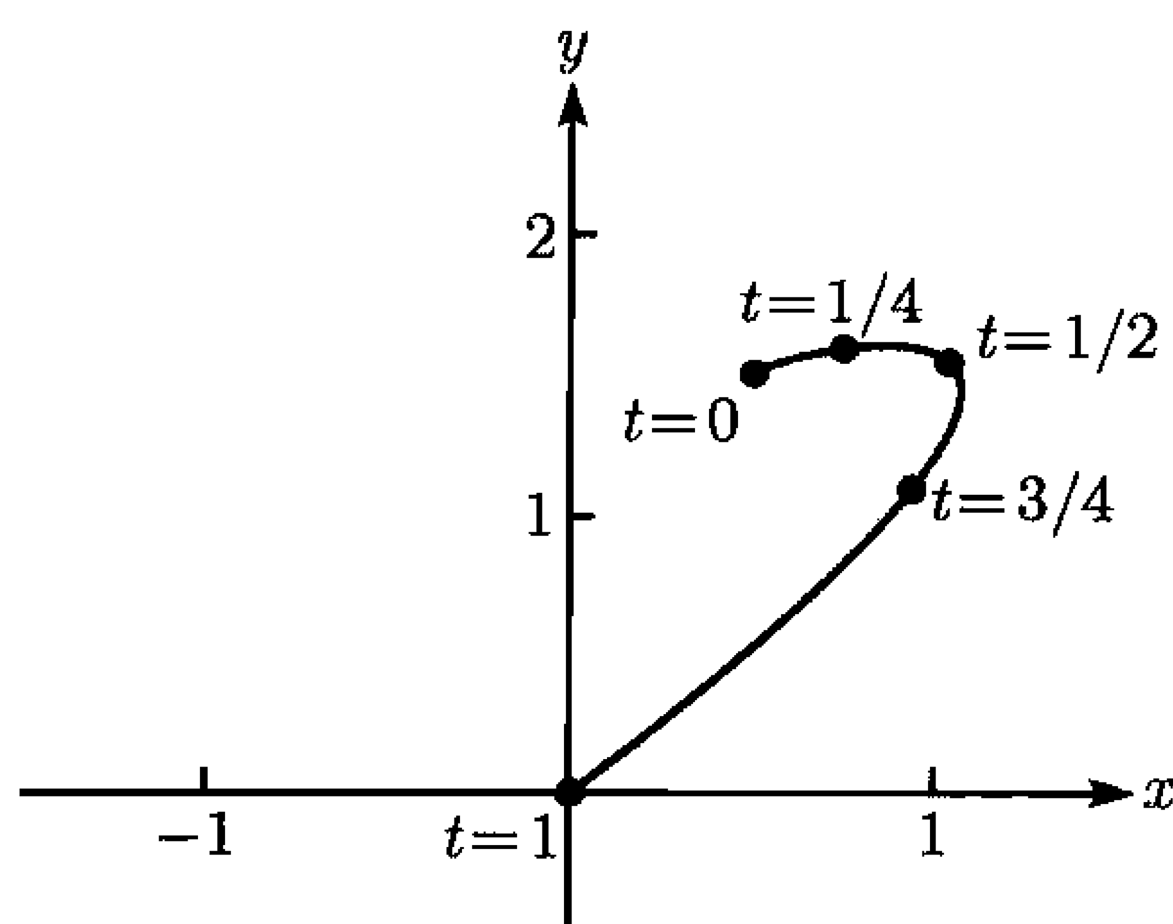


图 5-6　由 Bézier 样条给出的参数曲线: 一般地, 参数 t 的相等间隔并不把路径分成相等的长度

由微积分可知: 从 t_1 到 t_2 的路径弧长是

$$\int_{t_1}^{t_2}\sqrt{x'(t)^2+y'(t)^2}\mathrm{d}t.$$

这个积分很少产生的封闭形式的表达式. 因此, 通常用一种自适应求积技术来控制路径的参数化.

建议习题

1. 写出用自适应求积计算从 $t=0$ 到 $t=T$(给定 $T\leqslant 1$) 的弧长的 MATLAB函数.
2. 写一个程序, 使得对任一输入 s, 求沿着曲线的这种方法的 s 的参数 $t^*(s)$. 换言之, 从 $t=0$ 到 $t=t^*(s)$ 的弧长被从 $t=0$ 到 $t=1$ 的弧长相除应等于 s. 用对分法确定点 $t^*(s)$ 到 3 位准确小数, 什么函数被置为零? 应该用什么加括号区间起动对分法?
3. 把路径 n 等分. 取 $n=4$ 及 $n=20$. 给出与图 5-6 类似的图, 表明这种等分.
4. 用 Newton 方法在第 2 步中代替对分法, 并重复第 2 步和第 3 步. 需要什么样的导数? 初始估计的最优选择是什么? 经过这种替代, 计算时间是否减少?

附录 A 显示了 MABLAT中备有的动画命令. 例如命令

```
set(gca,'XLim',[-2 2],'YLim',[-2 2],'Drawmode','fast',...
   'Visible','on');
```

```
cla
axis square
ball=line('color','r','Marker','o','MarkerSize',10,...
   'LineWidth',2, 'erase','xor','xdata',[],'ydata',[]);
```

通过下面的命令定义了一个指定位置 (x, y) 的"球":

```
set(ball,'xdata',x,'ydata',y); drawnow;pause(0.01)
```

把这一行放到改变 x 和 y 的循环中, 使这个球沿着 MATLAB图像视窗中的路径移动.

5. 用 MATLAB动画命令显示沿着路径的移动, 开始以原参数 $0 \leqslant t \leqslant 1$ 的速度, 然后以由 $t^*(s)(0 \leqslant s \leqslant 1)$ 给出的 (恒定) 速度.
6. 等分你选取的路径进行实验. 例如在平面 (或在三维空间) 中选取 4 个点, 写出相应的 Bézier 样条, 并且把它分成 n 部分.
7. 写出路径 P 按照任意前进曲线 $C(x)(0 \leqslant s \leqslant 1)$ 移动的程序. 这里 $C(0) = 0$ 及 $C(1) = 1$. 目标是沿着曲线 C 以这样的方法移动: 路径总弧长的比 $C(x)$ 在 0 和 s 间移动. 例如, 沿着路径以恒定速度运动, 可以用 $C(s) = s$ 来表示. 尝试以前进曲线 $C(s) = s^{\frac{1}{3}}$ 及 $C(s) = s^2$ 为例.

关于平面和空间曲线的重新参数化的更多细节与应用, 可参阅 [1] 和 [5].

软件和进一步阅读

闭 Newton-Cotes 方法和开 Newton-Cotes 方法是近似计算定积分的基本工具. Romerg 积分是一种加速的形式. 大多数商用软件工具以某种形式包含自适应求积. 有关数值微分和数值积分的经典教科书包括 [2, 9, 7, 3, 4].

许多有效的求积技术通过 Netlib(**www.netlib.org/quadpack**) 中备有的公共流通软件包 Quadpack[8] 中的 Fortran 子程序来执行. Gauss-Kronrod 方法是基于 Gauss 求积法的一种自适应方法. Quadpack 分别提供了非自适应与自适应方法 QNG 和 QAG, 其中后者基于 Gauss-Kornod 方法. IMSL 和 NAG 中的程序都是基于 Quadpack 子程序. 例如在 IMSL 中的**quadrature**类是 java 实现.

MATLAB的**quad**命令是自适应复合 Simpson 求积的实现, 并且**dblquad**处理了二重积分. MATLAB的符号工具箱有命令**diff**和**int**, 它们分别用于符号微分和符号积分.

n 元函数的积分可以通过直接推广一维方法来进行, 只要积分区域是简单的, 例如, 见 [2,6]. 对于某些复杂区域, 需要用 Monte-Carlo 积分. Monte-Carlo 较易实现, 但是一般收敛更慢. 这些议题将在第 9 章中进一步讨论.

第6章 常微分方程

塔科马海峡大桥 (Tacoma Narrows Bridge) 曾经是世界上第三长的悬索桥, 以其在高速大风期间的明显垂直振动而闻名于世. 1940 年 11 月 7 日上午大约 11 点左右, 大桥断裂, 落入普吉特 (Puget) 海湾.

但是在断裂之前的运动却主要是左右扭转. 此前很难遇见, 在断裂前扭转持续了 45 分钟. 扭转运动最后变得大到足以折断支撑缆索从而使大桥迅速崩毁.

从那时起, 建筑师和工程师们对断裂的原因一直争论不休. 高速大风作为空气动力学的原因造成了垂直振动, 使大桥像机翼一样动作. 但是大桥整体并非因为严格的垂直运动而处于危险之中. 问题是, 扭转振动是如何产生的.

实例检验 6.4 节后的实例检验 6 中提出了一个微分方程模型, 它探索了扭转振动的可能的作用原理.

微分方程是含有导数的方程. 一阶微分方程

$$y'(t) = f(t, y(t))$$

表示量 y 关于目前时间和当前的量值的改变率. 微分方程组用于建模、推断和预测随时间而改变的系统.

大量有趣的方程没有封闭形式的解, 只能求助于近似解. 本章涉及借助计算方法的常微分方程的近似解. 引入微分方程的概念后, 我们要详细地叙述和分析 Euler 方法. 尽管 Euler 方法太简单, 在应用并不大量使用, 但是它是有关键性意义的, 因为这个主题的许多重要结果都能够从它非常简单的来龙去脉中容易地去理解.

随后是更复杂的高级方法, 并且讨论了有趣的微分方程组的例子. 为了有效地求解, 变步长是重要的, 并且对于刚性问题而言还需要用特殊的方法. 本章结束时将介绍隐式方法和多步方法.

6.1 初值问题

许多对自然现象成功建模的物理定律都是以微分方程的形式给出的. 牛顿把他的运动定律写成形式 $F = ma$, 它就把作用于物体的合力和该物体的加速度联系了起来. 这里的加速度是位移的二阶导数. 事实上, 牛顿定律的假定以及需要将它们写成微积分形式的基础的发展, 构成了科学史中非常重要的革命的一部分.

一个称为**逻辑斯谛方程**(logistic equation) 的简单模型把某种生物种群的总数的变化率建模为

$$y' = ay(1-y), \tag{6.1}$$

这里 y' 表示关于时间 t 的导数. 如果把 y 理解成总数关于这种动物生活环境的承载能力的比例, 那么我们预期 y 增长到接近这种承载能力后达到平衡. 微分方程 (6.1) 说明变化率 y' 与当前的总数与 "剩余的承载能力"$1-y$ 的乘积成比例. 因此, 无论当总数小 (y 接近 0) 时还是当总数接近承载能力 (y 接近 1) 时, 变化率都是小的.

常微分方程 (6.1) 一般情况下具有无穷多解. 通过给定初始条件, 可以在无穷多的解中确定我们感兴趣的那一个解 (6.12 节将更精确地阐述存在性和唯一性). 一阶常微分方程**初值问题**是求解在指定的区间 $a \leqslant t \leqslant b$ 上给定初始条件的方程

$$\begin{cases} y' = f(t,y), \\ y(a) = y_a, \\ t \in [a,b]. \end{cases} \tag{6.2}$$

像在图 6-1a 中一样, 把微分方程想象成一个斜度场是有益的. 可以把微分方程 (6.1) 看成对 (t,y) 的任一当前值确定一个斜度. 假如我们使用箭头画出在 ty 平面上的每一点处的斜度, 那么就得到了微分方程的**斜度场**(slope field) 或者**方向场**(direction field). 若方程的右端 $f(t,y)$ 不依赖于 t, 则这种方程是**自控的**(autonomous). 这在图 6-1 中是显然的.

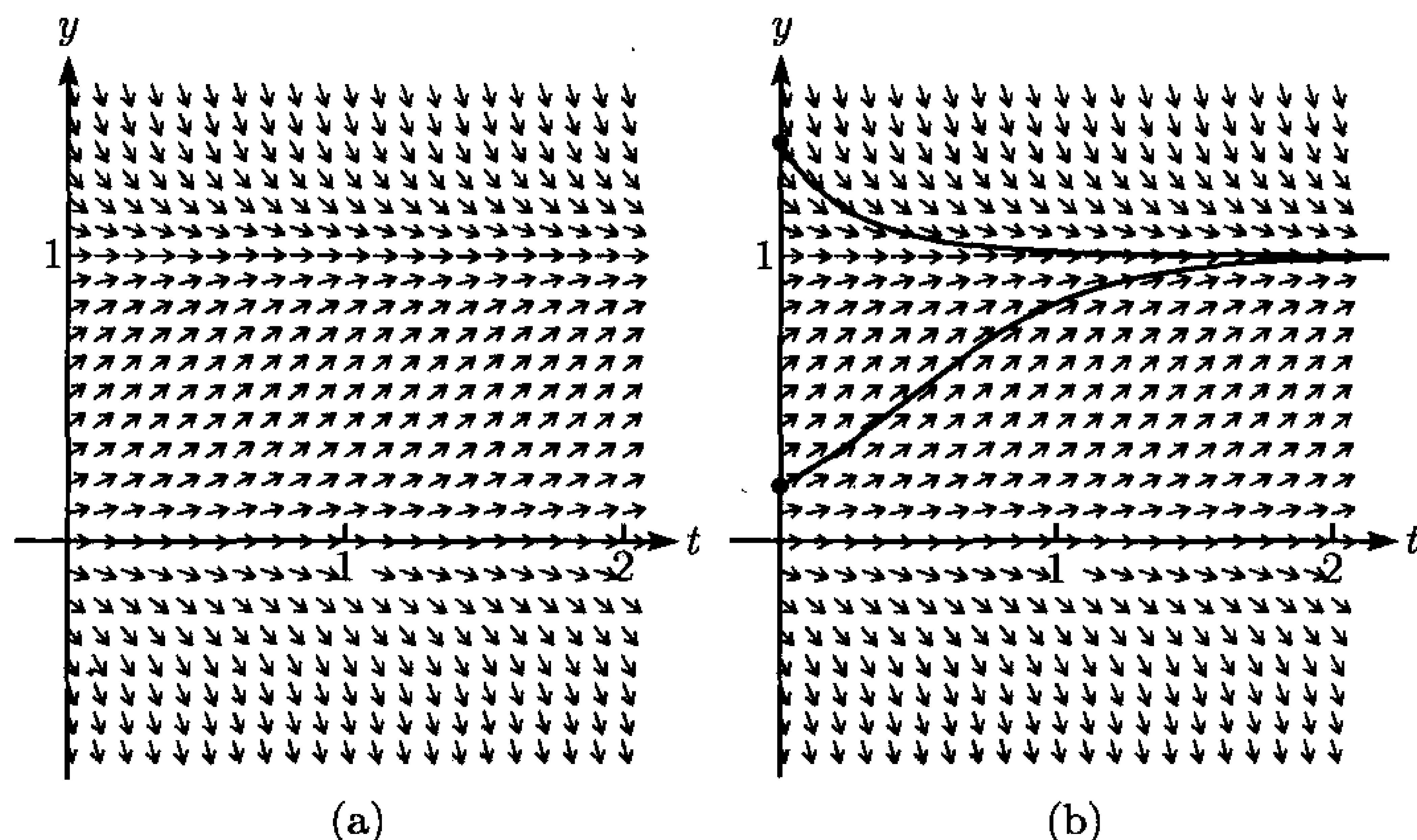


图 6-1　逻辑斯谛微分方程: (a) 自控方程的定义, 方向场沿 y 方向变化, 但是对所有的 t 是常数; (b) 微分方程的两个解

当在方向场中给定一个初始条件, 就能从无穷多个解中识别出一个解来. 在图 6-1b 中, 画出了分别开始于初值 $y(0) = 0.2$ 及 $y(0) = 1.4$ 的两个不同的解.

方程 (6.1) 有一个能表示成初等函数的解. 只要初始条件 $y_0 \neq 1$, 通过微分和代入, 就可以验证

$$y(t) = 1 - \frac{1}{1 + \frac{y_0}{1-y_0}e^{at}} \tag{6.3}$$

是初值问题

$$\begin{cases} y' = ay(1-y), \\ y(0) = y_0, \\ t \in [0, T] \end{cases} \tag{6.4}$$

的解, 该解沿着图 6-1b 中的箭头. 当 $y_0 = 1$ 时, 解就是 $y(t) = 1$, 它也能用同样的方法得到验证.

6.1.1 Euler 方法

逻辑斯谛方程有一个显式、相当简单的解. 更一般的情形是微分方程并不具有显式解. 图 6-1 中的几何形状提示了一种交错逼近: 通过沿着箭头来计算地 "求解" 微分方程. 从初始条件 (t_0, y_0) 开始, 然后沿着那里确定的方向; 在短距离移动后, 重新计算在新的点 (t_1, y_1) 处的斜率; 按新的斜率继续向前移动, 并且重复这个过程. 这个过程将产生一些误差, 这是因为在斜率的计算中, 不是沿着完全精确的斜度移动. 但是, 如果斜率变化较慢, 我们就能得到初值问题解的一个相当好的近似.

例 6.1 画出以下初值问题的方向场:

$$\begin{cases} y' = ty + t^3, \\ y(0) = y_0, \\ t \in [0, 1]. \end{cases} \tag{6.5}$$

图 6-2a 表示了其方向场. 在平面上的每一点 (t, y) 处, 画出斜率等于 $ty + y^3$ 的箭头. 因为 t 明显地出现在方程的右端, 所以这个初值问题是非自控的. 从方向场也很清楚地看到, 它随着 t 和 y 的不同而变化. 在初始条件 $y(0) = 1$ 时, 精确解 $y(t) = 3e^{\frac{t^2}{2}} - t^2 - 2$ 在图 6-2b 给出. 显式解的推导可见例 6.6. ◀

图 6-2b 说明一种沿着方向场进行计算的方法的执行过程. 这种方法称为 Euler 方法. 开始沿着 t 轴以等步长 h 给定 $n+1$ 个点

$$t_0 < t_1 < t_2 < \cdots < t_n$$

在图 6-2b 中, 我们选取 t 的值为

$$t_0 = 0.0, \quad t_1 = 0.2, \quad t_2 = 0.4, \quad t_3 = 0.6, \quad t_4 = 0.8, \quad t_5 = 1.0 \tag{6.6}$$

其步长 $h = 0.2$.

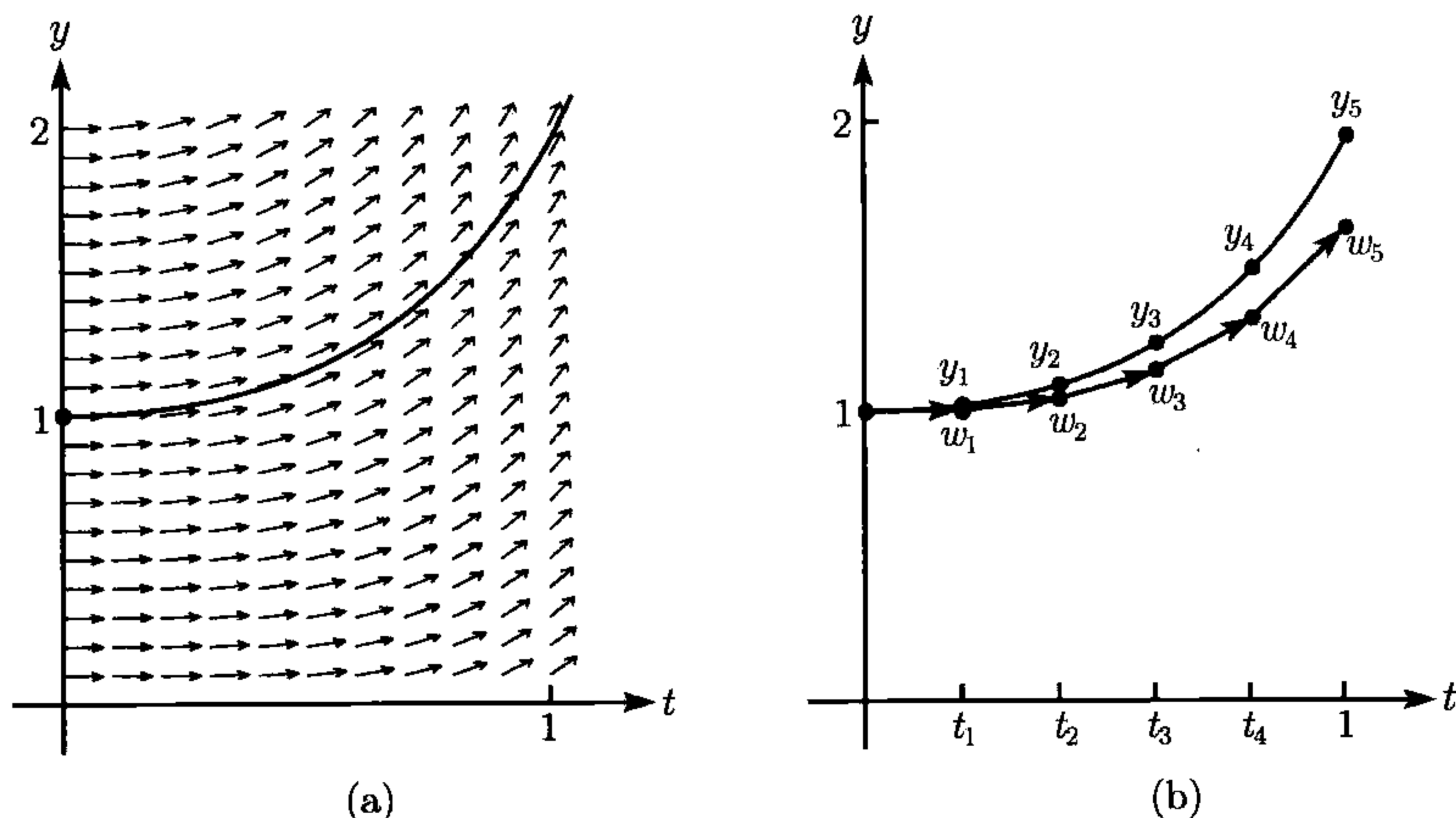


图 6-2 初值问题 (6.5) 的解: (a) 随时间 t 变化的非自控方程的方向场, 给出的解满足 $y(0) = 1$; (b) 取步长 $h = 0.2$, 对方程应用 Euler 方法

开始取 $w_0 = y_0$, 在每一个 t_i 处沿着方向场得到 t_{i+1} 处的近似

$$w_{i+1} = w_i + hf(t_i, w_i),$$

这是因为 $f(t_i, w_i)$ 表示解的斜率. 注意 y 的改变等于水平距离 h 乘以斜率. 如图 6-2b 所示, 每一个 w_i 是解在 t_i 处的近似.

这种方法的公式可以表示如下:

Euler 方法

$$\begin{aligned} w_0 &= y_0, \\ w_{i+1} &= w_i + hf(t_i, w_i). \end{aligned} \tag{6.7}$$

例 6.2 用 Euler 方法解初值问题 (6.5). 初始条件 $y_0 = 1$.

微分方程的右端 $f(t, y) = ty + y^3$. 因此,Euler 方法便是迭代

$$\begin{aligned} w_0 &= 1, \\ w_{i+1} &= w_i + h(t_i w_i + t_i^3). \end{aligned} \tag{6.8}$$

采用步长 $h = 0.2$ 的网格 (6.6), 从公式 (6.8) 迭代计算近似解. 由 Euler 方法给出的 w_i 值画在图 6-2b 中. 表 6-1 给出 w_i 与准确值 y_i 的比较, 也给出了每一步的误差 $e_i = |y_i - w_i|$. 尽管最大误差最终并不总是能求出, 但是误差将从初始条件处的 0 增大到区间端点处的最大值.

使用步长 $h = 0.1$ 的 Euler 方法使误差减小, 正如在图 6-3a 中看到的那样. 再用 (6.8), 计算得到表 6-2.

表 6-1

步骤	t_i	w_i	y_i	e_i
0	0.0	1.000 0	1.000 0	0.000 0
1	0.2	1.000 0	1.020 6	0.020 6
2	0.4	1.041 6	1.089 9	0.048 3
3	0.6	1.137 7	1.231 7	0.093 9
4	0.8	1.317 5	1.491 4	0.173 9
5	1.0	1.630 6	1.946 2	0.315 5

表 6-2

步骤	t_i	w_i	y_i	e_i
0	0.0	1.000 0	1.000 0	0.000 0
1	0.1	1.000 0	1.005 0	0.005 0
2	0.2	1.010 1	1.020 6	0.010 5
3	0.3	1.031 1	1.048 1	0.017 0
4	0.4	1.064 7	1.089 9	0.025 1
5	0.5	1.113 7	1.149 4	0.035 7
6	0.6	1.181 9	1.231 7	0.049 7
7	0.7	1.274 4	1.342 9	0.068 4
8	0.8	1.397 9	1.491 4	0.093 4
9	0.9	1.561 0	1.687 9	0.126 9
10	1.0	1.774 4	1.946 2	0.171 8

把对 $h = 0.1$ 进行计算得到的误差 e_{10} 与对 $h = 0.2$ 计算得到的误差 e_5 进行比较. 注意步长 h 减半的结果使在 $t = 1.0$ 处的误差大约也减小了一半. ◀

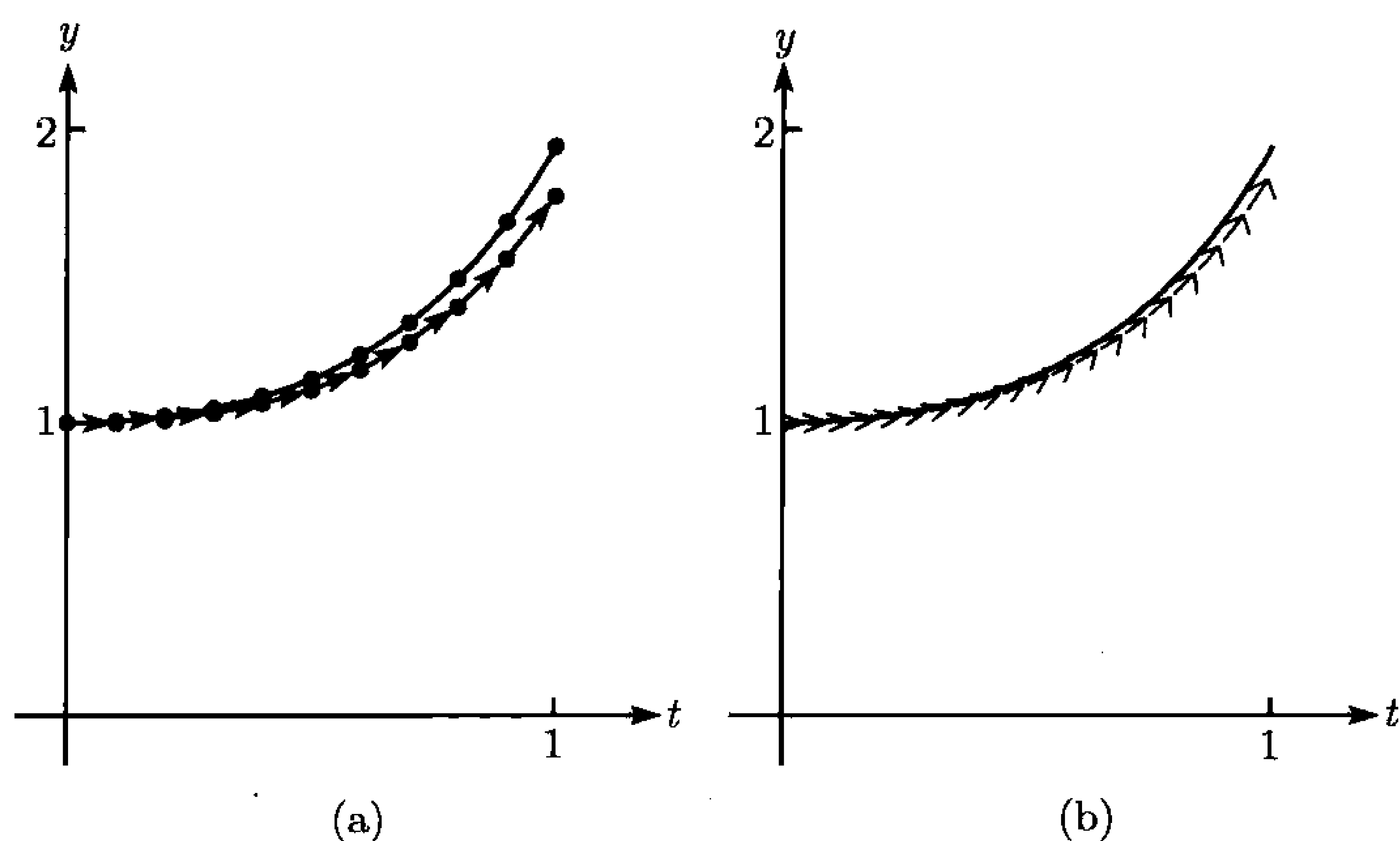


图 6-3　用于初值问题 (6.5) 的 Euler 方法: 除步长外, 确如在图 6-2 中那样, 这里的箭头表示 Euler 步. (a) 10 步, $h = 0.1$;　(b) 20 步,$h = 0.05$

在下面的 MATLAB code 中实现 Euler 方法, 它被写成 3 个模块. 绘图程序调用了执行每一 Euler 步的子程序, 而 Euler 步本身又调用了包含微分方程右端 f 的函数. 使用此形式, 以后既容易把右端交换成另一个微分方程, 又可把 Euler 方法交换成更一种更复杂的方法. 下面就是代码.

```
%Program 6.1 Euler's Method for Solving Initial Value Problems
%Use with ydot.m to evaluate rhs of differential equation
% Input: interval [a,b], initial value y0, step size h
% Output: time steps t, solution y
% Example usage: euler([0 1],1,0.1);
function [t,y]=euler(int,y0,h)
t(1)=int(1); y(1)=y0;
n=round((int(2)-int(1))/h);
for i=1:n
  t(i+1)=t(i)+h;
  y(i+1)=eulerstep(t(i),y(i),h);
end
plot(t,y)

function y=eulerstep(t,y,h)
%one step of Euler's Method
%Input: current time t, current value y,  stepsize h
%Output: approximate solution value at time t+h
y=y+h*ydot(t,y);

function z=ydot(t,y)
z=t*y+t^3;
```

在 $t=1$ 处, 把 (6.5) 的 Euler 方法近似解与精确解进行比较, 并把以前的结果从 $n=5$ 和 $n=10$ 扩大, 便得到表 6-3.

表 6-3

步数 n	步长 h	在 $t=1$ 处的误差
5	0.200 00	0.315 5
10	0.100 00	0.171 8
20	0.050 00	0.089 9
40	0.025 00	0.046 0
80	0.012 50	0.023 3
160	0.006 25	0.011 7
320	0.003 12	0.005 9
640	0.001 57	0.002 9

在表 6-3、图 6-3 和图 6-4 中, 有两个明显的事实. 首先是误差非零. 由于 Euler 方法取的步数不是无穷小, 斜率沿着每一步而变化, 并且近似解并不精确地落在解

曲线上. 其次是误差随着步长的减小而减小, 如在图 6-3 看到的. 从表 6-3 中可看到误差与 h 成比例, 这一事实将在下一节中得到证实.

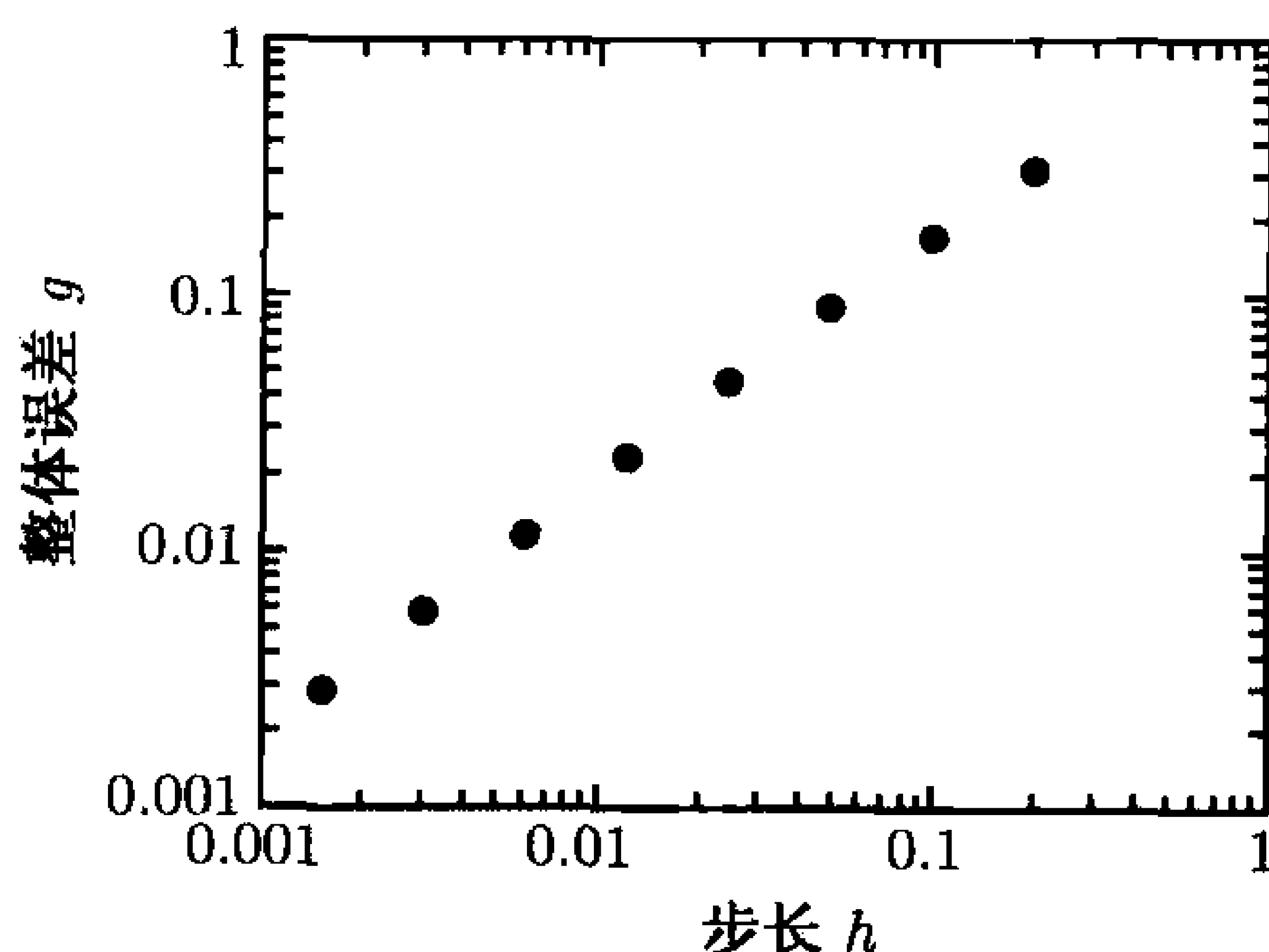


图 6-4 作为 Euler 方法的步长函数的误差: 在双对数坐标图中, 在 $t = 1$ 处 (6.5) 的近似解与准确解的差的斜率为 1. 因此, 对于小的 h, 误差正比于步长 h

例 6.3 对初值问题

$$\begin{cases} y' = cy, \\ y(0) = y_0, \\ t \in [0, 1], \end{cases} \tag{6.9}$$

寻找其 Euler 方法的公式.

对于 $f(t, y) = cy$, 这里 c 是常数,Euler 方法给出

$$w_0 = y_0,$$
$$w_{i+1} = w_i + hcw_i = (1 + hc)w_i, \qquad i = 1, 2, 3, \cdots. \quad ◀$$

用**分离变量**(separation of variable) 法可以求得方程 $y' = cy$ 的精确解. 假定 $y \neq 0$, 两边除以 y, 分离变量, 再积分:

$$\frac{\mathrm{d}y}{y} = c\mathrm{d}t,$$
$$\ln|y| = ct + k,$$
$$|y| = \mathrm{e}^{ct+k} = \mathrm{e}^k \mathrm{e}^{ct}.$$

初始条件 $y(0) = y_0$ 意味着 $y = y_0\mathrm{e}^{ct}$.

就这种简单情形, 可以证明当步数 $n \to \infty$ 时, Euler 方法的近似解收敛于准确解. 注意到

$$w_i = (1 + hc)w_{i-1} = (1 + hc)^2 w_{i-2} = \cdots = (1 + hc)^i w_0.$$

对固定的 t, 置步长 $h=\frac{t}{n}$(n 是整数), 于是在 t 处的近似值是

$$w_n=(1+hc)^n y_0=\left(1+\frac{ct}{n}\right)^n y_0.$$

古典的极限公式告诉我们,

$$\lim_{n\to\infty}\left(1+\frac{ct}{n}\right)^n=\mathrm{e}^{ct}$$

这就证明了当 $n\to\infty$ 时, Euler 公式的近似解收敛于准确解.

6.1.2 解的存在性、唯一性和连续性

本节提供计算初值问题方法的某些理论基础. 在开始计算一个问题的解之前, 了解以下两点是有帮助的: (1) 解存在; (2) 仅有一个解. 因此算法不会因计算哪一个解而混淆. 在适当情形下, 初值问题恰有一解.

定义 6.1 *假定存在常数 L(称为**Lipschitz 常数**), 使得函数 $f(t,y)$ 对 $S=[a,b]\times[y_1,y_2]$ 中的任意 $(t,y_1),(t,y_2)$ 都有*

$$|f(t,y_1)-f(t,y_2)|\leqslant L|y_1-y_2|,$$

*则称函数 $f(t,y)$ 在矩形 S 中关于变量 y 满足**Lipschitz**条件.*

一个关于 y 满足 Lipschitz 条件的函数关于 y 是连续的, 但未必可微.

例 6.4 对 (6.5) 式中的右端 $f(t,y)=ty+t^3$, 求 Lipschitz 常数.

在集合 $0\leqslant t\leqslant 1,-\infty<y<\infty$ 上, 函数 $f(t,y)=ty+t^3$ 关于变量 y 满足 Lipschitz 条件. 检查发现

$$|f(t,y_1)-f(t,y_2)|=|ty_1-ty_2|\leqslant|t||y_1-y_2|\leqslant|y_1-y_2| \tag{6.10}$$

在上述集合内成立. 于是 Lipschitz 常数 $L=1$. ◀

尽管定义 6.1 强调集合 S 是矩形, 但是更一般地, S 可以是一个**凸集**, 即能包含联结该集合内任意两点的线段的集合. 假如函数 f 关于变量 y 连续可微, 偏导数 $\frac{\partial f}{\partial y}$ 的绝对值的最大值就是 Lipschitz 常数. 根据中值定理, 对每一个固定的 t, 在 y_1 与 y_2 之间的存 c, 使得

$$\frac{f(t,y_1)-f(t,y_2)}{y_1-y_2}=\frac{\partial f}{\partial y}(t,c).$$

因此, L 可取为 $\left|\frac{\partial f}{\partial y}(t,c)\right|$ 在该集合内的最大值.

Lipschitz 假设保证了初值问题解的存在性和唯一性. 定理 6.2 的证明可参考 [3].

定理 6.2 在集合 $[a,b]\times[y_1,y_2]$ 上, 若 $f(t,y)$ 关于变量 y 满足 Lipschitz 条件, 那么在 a 和 b 之间存在 c, 使得初值问题

$$\begin{cases} y' = f(t,y), \\ y(a) = y_a, \\ t \in [a,c] \end{cases} \tag{6.11}$$

只有一个解 $y(t)$, 而且, 如果 f 在 $[a,b]\times(-\infty,\infty)$ 上是 Lipschitz 型的, 那么在区间 $[a,b]$ 上只存在一个解.

理解定理 6.2 的保留条件 (fine print) 是重要的, 特别是想要数值计算这个解时. 初值问题在 $[a,b]\times[y_1,y_2]$ 上满足 Lipschitz 条件这一事实并不保证在整个区间$[a,b]$ 上对于 t 有一个解. 简单的理由是解可能偏离到 y 的范围 $[y_1,y_2]$ 之外, 而 Lipschitz 常数在 $[y_1,y_2]$ 上有效. 我们能够说的最好结果是在某个较短的区间 $[a,c]$ 存在解. 下面的例子阐述了这一点.

例 6.5 在哪个区间 $[0,c]$ 上, 下列初值问题有唯一解?

$$\begin{cases} y' = y^2, \\ y(0) = 1, \\ t \in [0,2]. \end{cases} \tag{6.12}$$

f 关于 y 的偏导数是 $2y$. 在集合 $0\leqslant t\leqslant 2, -10\leqslant y\leqslant 10$ 中,Lipshcitz 常数 $\max|2y| = 20$. 定理 6.2 保证解从 $t=0$ 开始, 并存在于某个区间 $[a,c](c>0)$ 上, 但是并不保证解在整个区间 $[0,2]$ 上.

事实上, 通过分离变量可求得微分方程 (6.12) 的唯一解是 $y(t)=\frac{1}{1-t}$. 当 t 趋于 1 时, 这个解趋于无穷. 换言之, 对任意的 $0<c<1$, 解存在于区间 $0\leqslant t\leqslant c$, 但是对于更大的 c, 解并不存在. 这个例子说明了定理 6.2 中 c 的作用. Lipschitz 常数 20 对 $|y|\leqslant 10$ 有效, 但是在 t 到达 2 之前, 解 y 已超出了 10. ◀

亮点 条件作用

在第 1 章和第 2 章中, 我们把误差的放大详述为量化输入信息的微小改变或误差对解的影响的一种方法. 定理 6.3 对初值问题的这一类似问题作出了细致的回答. 当初始条件 (输入信息)$Y(a)$ 改变成 $Z(a)$, t 时间单位后, 输出的最大可能改变 $Y(t)-Z(t)$, 关于 t 是指数型的而关于初始条件的差是线性的. 后者意味着, 对固定的时间 t, 我们能够说 "条件数" 等于 $e^{L(t-a)}$.

定理 6.3 说明了常微分方程的稳定性 (误差的放大) 的基本情形. 假如对于微分方程右端而言存在 Lipschitz 常数, 那么以后的解是带有新的 Lipschitz 常数的关

于初值的 Lipschitz 函数, 这个新的 Lipschitz 常数是原来的那个 (Lipschitz 常数) 的指数函数. 这是 Gronwall 不等式的一种形式.

定理 6.3 在 $S=[a,b]\times[y_1,y_2]$ 上, 若 $f(t,y)$ 关于变量 y 是 Lipschitz 函数. 又假设 $Y(t)$ 和 $Z(t)$ 是微分方程

$$y'=f(t,y)$$

在 S 中的分别满足初始条件 $Y(a)$ 和 $Z(a)$ 的两个解, 那么

$$|Y(t)-Z(t)|\leqslant e^{L(t-a)}|Y(a)-Z(a)|. \tag{6.13}$$

证 如果 $Y(a)=Z(a)$, 则根据唯一性,$Y(t)=Z(t)$, 那么 (6.13) 显然满足. 我们再假设 $Y(a)\neq Z(a)$, 为了避免与唯一性矛盾, 此时 $Y(t)\neq Z(t)$ 对区间中所有的 t 都成立.

定义 $u(t)=Y(t)-Z(t)$. 由于 $u(t)$ 或者严格为正或者严格为负, 又因为 (6.13) 仅与 $|u|$ 有关, 所以可以假设 $u>0$. 于是 $u(a)=Y(a)-Z(a)$, 而且其导数 $u'(t)=Y'(t)-Z'(t)=f(t,Y(t))-f(t,Z(t))$. Lipschitz 条件意味着

$$u'=|f(t,Y)-f(t,Z)|\leqslant L|Y(t)-Z(t)|=L|u(t)|=Lu(t).$$

因此, $(\ln u)'=\frac{u'}{u}\leqslant L$. 根据中值定理得到

$$\frac{\ln u(t)-\ln u(a)}{t-a}\leqslant L,$$

这可简化为

$$\ln\frac{u(t)}{u(a)}\leqslant L(t-a), u(t)\leqslant u(a)e^{L(t-a)}.$$

这就是所要的结果.

回到例题 6.4, 定理 6.3 意味着开始于不同初值的两个解 $Y(t)$ 和 $Z(t)$, 绝不会比乘积因子 $e^t(0\leqslant t\leqslant 1)$ 分离得更快. 事实上, 初值 Y_0 处的解是 $Y(t)=(2+Y_0)e^{\frac{t^2}{2}}-t^2-2$, 所以上述两个解的差是

$$\begin{aligned}|Y(t)-Z(t)|&\leqslant\left|(2+Y_0)e^{t^2/2}-t^2-2-((2+Z_0)e^{t^2/2}-t^2-2)\right|\\&\leqslant|Y_0-Z_0|e^{t^2/2},\end{aligned} \tag{6.14}$$

它小于 $|Y_0-Z_0|e^t(0\leqslant t\leqslant 1)$, 恰如定理 6.3 所示.

6.1.3 一阶线性方程

一类特殊的容易求解的常微分方程提供了一类可以说明问题的例子. 它们是右端关于 y 是线性的一阶方程. 考虑初值问题

$$\begin{cases} y' = g(t)y + h(t), \\ y(a) = y_a, \\ t \in [a, b]. \end{cases} \tag{6.15}$$

首先注意, 如果 $g(t)$ 在 $[a, b]$ 上连续, 则由定理 6.2, 取 $L = \max\limits_{[a,b]} g(t)$ 为 Lipschitz 常数, 方程存在唯一解. 通过把方程乘以"积分因子"这种技巧, 可以求出方程的解.

积分因子是 $\mathrm{e}^{-\int g(t)\mathrm{d}t}$. 用它乘以方程两边, 得到

$$\begin{aligned} (y' - g(t)y) - \mathrm{e}^{-\int g(t)\mathrm{d}t} &= \mathrm{e}^{-\int g(t)\mathrm{d}t} h(t), \\ (y\mathrm{e}^{-\int g(t)\mathrm{d}t})' &= \mathrm{e}^{-\int g(t)\mathrm{d}t} h(t), \\ y\mathrm{e}^{-\int g(t)\mathrm{d}t} &= \int \mathrm{e}^{-\int g(t)\mathrm{d}t} h(t)\mathrm{d}t. \end{aligned}$$

解得

$$y(t) = \mathrm{e}^{\int g(t)\mathrm{d}t} \int \mathrm{e}^{-\int g(t)\mathrm{d}t} h(t)\mathrm{d}t. \tag{6.16}$$

如果积分因子能简单地表示出来, 用这种方法就能得到一阶线性方程 (6.15) 的显式解.

例 6.6 求解一阶线性微分方程

$$\begin{cases} y' = ty + t^3, \\ y(0) = y_0. \end{cases} \tag{6.17}$$

积分因子是

$$\mathrm{e}^{-\int g(t)\mathrm{d}t} = \mathrm{e}^{-\frac{t^2}{2}}.$$

根据 (6.16), 解是

$$\begin{aligned} y(t) &= \mathrm{e}^{\frac{t^2}{2}} \int \mathrm{e}^{-\frac{t^2}{2}} t^3 \mathrm{d}t = \mathrm{e}^{\frac{t^2}{2}} \int \mathrm{e}^{-u}(2u)\mathrm{d}u \\ &= 2\mathrm{e}^{\frac{t^2}{2}} \left[-\frac{t^2}{2}\mathrm{e}^{-\frac{t^2}{2}} - \mathrm{e}^{-\frac{t^2}{2}} + C \right] = -t^2 - 2 + 2C\mathrm{e}^{\frac{t^2}{2}}, \end{aligned}$$

这里进行了变量代换 $u = \frac{t^2}{2}$. 求解积分常数 C, 得到 $y_0 = -2 + 2C$, 所以 $C = (2 + y_0)/2$. 因此

$$y(t) = (2 + y_0)\mathrm{e}^{\frac{t^2}{2}} - t^2 - 2.$$

◀

习题 6.1

1. 证明函数 $y(t) = t\sin t$ 是以下微分方程的解:
   (a) $y + t^2\cos t = ty'$; (b) $y'' = 2\cos t - y$; (c) $t(y'' + y) = 2y' - 2\sin t$.
2. 证明函数 $y(t) = \mathrm{e}^{\sin t}$ 是以下初值问题的解:
   (a) $y' = y\cos t, y(0) = 1$; (b) $y'' = (\cos t)y' - (\sin t)y, y(0) = 1, y'(0) = 1$;
   (c) $y'' = y(1 - \ln y) - (\ln y)^2), y(\pi) = 1, y'(\pi) = -1$.
3. 给定 $y(0) = 1$, 用分离变量法求以下微分方程初值问题的解:
   (a) $y' = t$; (b) $y' = t^2y$; (c) $y' = 2(t + 1)y$;
   (d) $y' = 5t^4y$; (e) $y' = \dfrac{1}{y^2}$; (f) $y' = \frac{t^3}{y^2}$.
4. 给定 $y(0) = 0$, 求以下一阶线性微分方程初值问题的解:
   (a) $y' = t + y$; (b) $y' = t - y$; (c) $y' = 4t - 2y$.
5. 取步长 $h = \frac{1}{4}$, 把 Euler 方法用于习题 3, 在区间 $[0, 1]$ 上解初值问题. 列出 $w_i, i = 0, 1, \cdots, 4$, 并且通过与准确解比较, 求在 $t = 1$ 处的误差.
6. 取步长 $h = \frac{1}{4}$, 把 Euler 方法用于习题 4, 在区间 $[0, 1]$ 上解初值问题. 通过与准确解比较, 求在 $t = 1$ 处的误差.
7. 定理 6.2 保证在 $[0, 1]$ 上的这些初值问题中哪些存在唯一解? 如果 Lipschitz 常数存在, 把它们求出来.(a) $y' = t$, (b) $y' = y$, (c) $y' = -y$, (d) $y' = -y^3$.
8. 画出习题 7 中微分方程的方向场, 并且画出开始于初始条件 $y(0) = 1, y(0) = 0$ 及 $y(0) = -1$ 的解的近似值的草图.
9. 求习题 7 中初值问题的解. 对每个方程使用习题 7 中的 Lipschitz 常数, 假如可能的话, 证明对于带有初始条件 $y(0) = 0$ 及 $y(0) = 1$ 的两个解, 满足定理 6.3 中的不等式.
10. (a) 假如 $a \neq 0$, 证明初值问题 $y' = ay + b, y(0) = y_0$ 的解是 $y(t) = \left(\frac{b}{a}\right)(\mathrm{e}^{at} - 1) + y_0\mathrm{e}^{at}$, (b) 证明分别带有初值 y_0 及 z_0 的解 $y(t), z(t)$ 满足定理 3 中的不等式.
11. 用分离变量法解初值问题 $y' = y^2, y(0) = 1$.
12. 求初值问题 $y' = ty^2, y(0) = 1$ 的解. 解存在的最大区间 $[0, b)$ 是什么?

计算机问题 6.1

1. 在 $[0, 1]$ 上, 取步长 $h = 0.1$, 对习题 3 中的初值问题, 用 Euler 方法求解. 打印出 t 值、Euler 近似解及每一步的误差 (与准确解的差).
2. 取步长 $h = 0.1, 0.05, 0.025$, 画出习题 3 在 [0,1] 中的初值问题的 Euler 方法近似解和真解.
3. 取步长 $h = 0.1, 0.05, 0.025$, 画出习题 4 在 [0,1] 中的初值问题的 Euler 方法近似解和真解.
4. 对习题 3 中的初值问题, 把 Euler 方法在 $t = 1$ 处的误差画作 $h = 0.1 \times 2^k (0 \leqslant k \leqslant 5)$ 的函数. 使用图 6-4 中的双对数坐标图.
5. 对习题 4 中的初值问题, 把 Euler 方法在 $t = 1$ 处误差画作 $h = 0.1 \times 2^k (0 \leqslant k \leqslant 5)$ 的函数.

6.2 初值问题解法分析

图 6-4 表明, 对习题 6.1, Euler 方法近似解的误差随着步长的递减而一致递减. 这在一般情况下都正确吗? 可以通过减小步长使误差小到我们所想的那样吗? 仔细研究 Euler 方法的误差将说明在一般情况下初值问题解法的结果.

6.2.1 局部截断误差和整体截断误差

图 6-5 给出了在解形如

$$\begin{cases} y' = f(t,y), \\ y(a) = y_a, \\ t \in [a,b] \end{cases} \tag{6.18}$$

的初值问题时, 像 Euler 方法那样的解法的一步示意图.

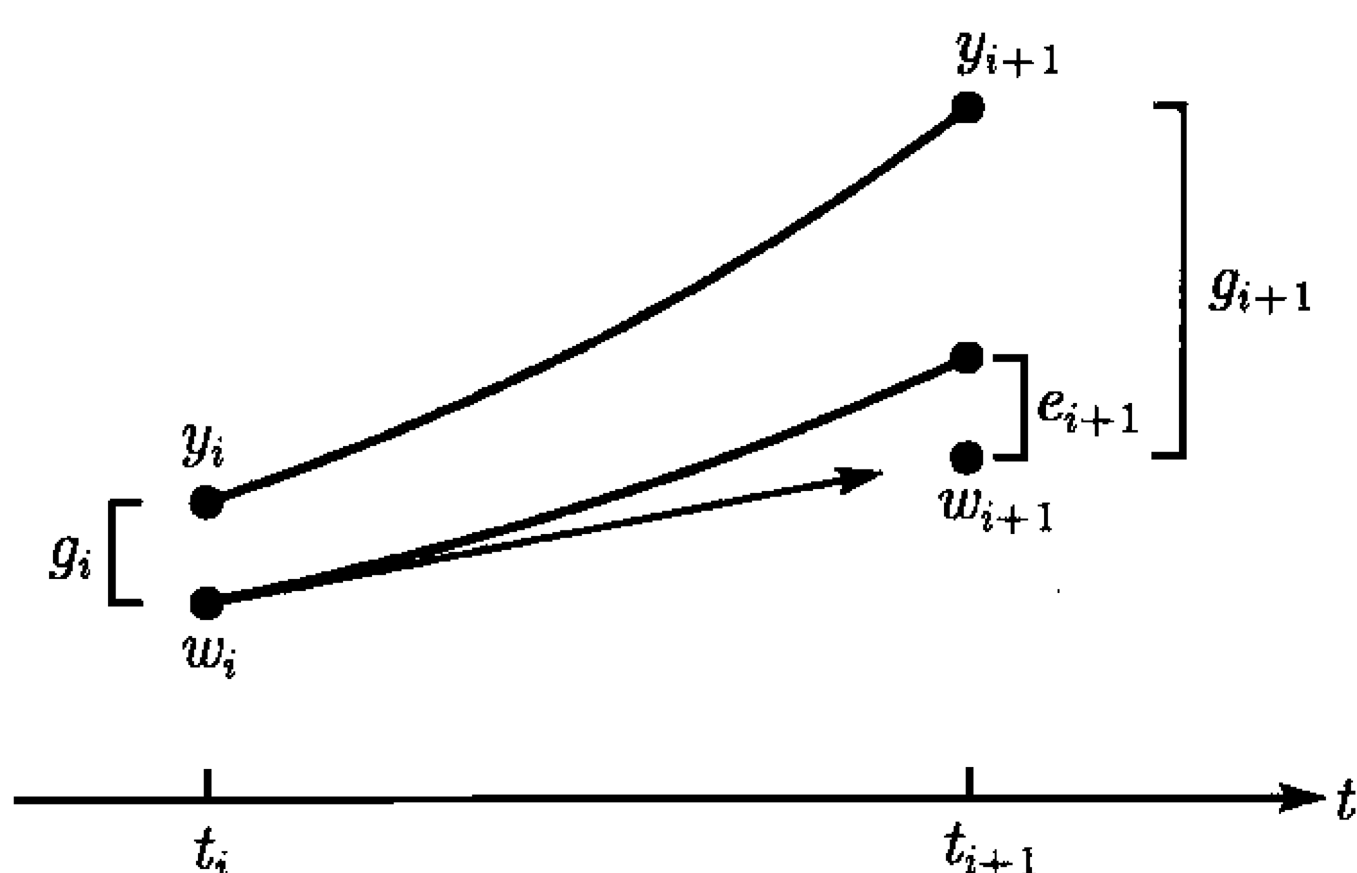


图 6-5 常微分方程解法的一步. Euler 方法在当前点按向量场的斜率沿着直线段到下一个点 (t_{i+1}, w_{i+1}). 上边的曲线代表微分方程的真解. 整体截断误差 g_{i+1} 是局部截断误差 e_{i+1} 以及以前各步积累和放大的误差之和

在第 i 步, 以前各步积累的误差被一起携带甚至会被放大, 而来自 Euler 近似的新误差还要加上. 为了更精确地阐述, 定义**整体截断误差**(global truncation error)

$$g_i = |w_i - y_i|$$

是常微分方程解法 (譬如 Euler 方法) 近似解和初值问题准确解之间的差. 同样, 定义**局部截断误差**(local truncation error), 或一步误差为

$$e_{i+1} = |w_{i+1} - z(t_{i+1})|, \tag{6.19}$$

它是在区间上解的值和 "一步初值问题"

$$\begin{cases} y' = f(t,y), \\ y(t_i) = w_i, \\ t \in [t_i, t_{i+1}] \end{cases} \tag{6.20}$$

的准确解之间的差.

(我们称解为 z, 是因为 y 已用来作为开始于准确的初始条件 $y(t_i)=y_i$ 的同样的初值问题的解). 局部截断误差是一步中出现的误差, 这一步是把前面解的近似值 w_i 作为出发点. 整体截断误差是由前面 i 步积累起来的误差. 图 6-5 说明了局部截断误差和整体截断误差. 每一步新的整体误差是以前各步放大了的整体误差和新的局部误差的结合. 因为放大, 所以整体误差不仅仅是局部误差之和.

例 6.7 求 Euler 方法的局部截断误差.

按照定义, 这是 Euler 方法在一步产生的新误差. 假定前面步 w_i 是准确的, 准确地解出初值问题 (6.20), 并把准确解 $y(t_{i+1})$ 与 Euler 方法近似解比较.

假设 y'' 连续, 在 $t_{i+1}=t_i+h$ 处的准确解是

$$y(t_i+h)=y(t_i)+hy'(t_i)+\frac{h^2}{2}y''(c).$$

根据 Taylor 定理, 上式对某个未知的 $c(t_i<c<t_{i+1})$ 成立. 因为 $y(t_i)=w_i$ 及 $y'(t_i)=f(t_i,w_i)$, 所以上式可以写成

$$y(t_{i+1})=w_i+hf(t_i,w_i)+\frac{h^2}{2}y''(c).$$

同时,Euler 方法表明

$$w_{i+1}=w_i+hf(t_i,w_i).$$

两式相减得到局部截断误差

$$e_{i+1}=|w_{i+1}-y(t_{i+1})|=\frac{h^2}{2}|y''(c)|,$$

它对区间中的某个 c 成立. 假设 M 是 y'' 在 $[a,b]$ 上的上界, 那么局部截断误差满足

$$e_i\leqslant\frac{Mh^2}{2}.$$ ◀

现在研究局部误差如何加到整体误差上去. 在初始条件 $y(a)=y_a$ 处, 整体误差是 $g_0=|w_0-y_0|=|y_a-y_a|=0$. 一步之后, 以前的步没有积累误差, 因此, 整体误差等于第一个局部误差,$g_1=e_1=|w_1-y_1|$. 两步之后, g_2 分解成局部截断误差加上来自以前步中积累的误差, 如在图 6-5 中所示. 定义 $z(t)$ 是初值问题

$$\begin{cases} y'=f(t,y), \\ y(t_1)=w_1, \\ t\in[t_1,t_2] \end{cases} \tag{6.21}$$

的解. 因此 $z(t_2)$ 是开始于初始条件 (t_1,w_1) 的解的准确值. 注意到, 假如用初始条件 (t_1,y_1), 我们将会得到 y_2, 它不同于 $z(t_2)$, 因为它落在实际的解曲线上. 于是

$e_2 = |w_2 - z(t_2)|$ 是在 $i = 2$ 步的局部截断误差. 另一个差 $|z(t_2) - y_2|$ 由定理 6.3 解释, 这是因为它是相同微分方程带有不同初始条件 w_1 和 g_1 的两个解的差. 因此

$$\begin{aligned} g_2 &= |w_2 - y_2| = |w_2 - z(t_2) + z(t_2) - y_2| \\ &\leqslant |w_2 - z(t_2)| + |z(t_2) - y_2| \\ &\leqslant e_2 + \mathrm{e}^{Lh} g_1 = e_2 + \mathrm{e}^{Lh} \mathrm{e}_1. \end{aligned}$$

对 $i = 3$ 步, 同上可得

$$g_3 = |w_2 - y_3| \leqslant e_3 + \mathrm{e}^{Lh} g_2 \leqslant e_3 + \mathrm{e}^{Lh} e_2 + \mathrm{e}^{2Lh} e_1. \tag{6.22}$$

同样, 在 i 步的整体截断误差满足

$$g_i = |w_i - y_i| \leqslant e_i + \mathrm{e}^{Lh} e_{i-1} + \mathrm{e}^{2Lh} e_{i-2} + \cdots + \mathrm{e}^{(i-1)Lh} e_1. \tag{6.23}$$

在例 6.7 中, 我们得到 Euler 方法有正比于 h^2 的局部截断误差. 更一般地, 若局部截断误差

$$e_i \leqslant Ch^{k+1}$$

对某个整数 k 及常数 $C > 0$ 成立, 那么

$$\begin{aligned} g_i &\leqslant Ch^{k+1}(1 + \mathrm{e}^{Lh} + \cdots + \mathrm{e}^{(i-1)Lh}) = Ch^{k+1} \frac{\mathrm{e}^{iLh} - 1}{\mathrm{e}^{Lh} - 1} \\ &\leqslant Ch^{k+1} \frac{\mathrm{e}^{L(t_i - a)} - 1}{Lh} = \frac{Ch^k}{L}(\mathrm{e}^{L(t_i - a)} - 1). \end{aligned} \tag{6.24}$$

注意局部截断误差是如何与整体截断误差相联系的. 对某个 k, 局部截断误差正比于 h^{k+1}. 粗略地讲, 整体截断误差把许多步的局部截断误差加在一起, 而步数正比于 h^{-1}, 即步长的倒数. 因此, 整体截断误差正比于 h^k. 这是前面计算的主要结论, 如定理 6.4 所述.

定理 6.4 若 $f(t,y)$ 关于变量 y 有 Lipschitz 常数 L, 并且初值问题 (6.2) 在 t_i 处的解的值 y_i 用 w_i 来近似,w_i 来自局部截断误差 $e_i \leqslant Ch^{k+1}$ 的单步常微分方程解法, 其中 C 是某个常数,$k \geqslant 0$. 那么, 对每个 $a < t_i < b$, 这个解法有整体截断误差

$$g_i = |w_i - y_i| \leqslant \frac{Ch^k}{L}(\mathrm{e}^{L(t_i - a)} - 1). \tag{6.25}$$

亮点　收敛性

定理 6.4 是关于单步微分方程解法的收敛性的主要定理, 整体误差对 h 的依赖性表明, 我们能够期望当 h 减小时, 误差也减小, 因此 (至少在确

定的算法中) 误差可以如我们所希望的那样小. 这就把我们带到另一个重点: 整体误差关于 h 的指数型依赖. 当时间增长时, 整体误差界可能变得非常大. 对于大的 t_i, 为了使整体误差小, 导致所需要的步长 h 可能太小而实际上难以做到.

当 $h \to 0$ 时, 若常微分方程的解法满足 (6.25), 就说这种方法具有 k**阶**(order). 例 6.7 说明 Euler 方法的局部截断误差的大小以 $\frac{Mh^2}{2}$ 为上界, 所以 Euler 方法的阶是 1. 对 Euler 方法情形的定理重新叙述就给出以下推论:

推论 6.5(Euler 方法的收敛性) *设 $f(t,y)$ 对变量 y 有 Lipschitz 常数 L, 初值问题 (6.2) 在 t_i 处的解 y_i 用 Euler 方法被 w_i 所近似. 令 M 是 $|y''(t)|$ 在 $[a,b]$ 的一个上界. 那么*

$$|w_i - y_i| \leqslant \frac{Mh}{2L}(\mathrm{e}^{L(t_i-a)} - 1). \tag{6.26}$$

例 6.8 求用于例 6.1 的 Euler 方法的误差界.

$[0,1]$ 上的 Lipschitz 常数是 $L = 1$, 既然解 $y(t) = 3\mathrm{e}^{\frac{t^2}{2}} - t^2 - 2$ 是已知的, 那么它的二阶导数确定为 $y''(t) = (t^2+2)\mathrm{e}^{\frac{t^2}{2}} - 2$, 它在 $[0,1]$ 上绝对值的上界是 $M = y''(1) = 3\sqrt{\mathrm{e}} - 2$. 推论 6.5 意味着在 $t = 1$ 处的整体截断误差必定小于

$$\frac{Mh}{2L}\mathrm{e}^L(1-0) = \frac{(3\sqrt{\mathrm{e}}-2)}{2}eh \approx 4.004h. \tag{6.27}$$

实际整体误差证实了这个上界, 如图 6-4 所示, 对小的 h, 它大致是 h 的两倍. ◀

到目前为止,Euler 方法似乎是十分简单安全的. 它的构造是直观的, 根据推论 6.5 产生的误差随着步长减小而变小. 然而, 对更困难的初值问题, Euler 方法很少使用. 还有更复杂的方法, 它们的阶, 或者 (6.25) 中 h 的幂次, 大于 1, 正像我们将看到的, 这就大大减小了整体误差. 我们以一个简单的例子来结束这一节, 在该例中需要减小误差.

例 6.9 把 Euler 方法用于初值问题

$$\begin{cases} y' = -4t^3y^2, \\ y(-10) = \dfrac{1}{10\,001}, \\ t \in [-10, 0]. \end{cases} \tag{6.28}$$

通过代入, 容易验证准确解是 $y(t) = \frac{1}{t^4+1}$. 这个解在我们关心的区间中令人满意. 我们将评估 Euler 方法逼近 $t = 0$ 处的解的能力.

图 6-6 表示 Euler 方法对解的近似, 其步长从底到顶分别是 $h = 10^{-3}, 10^{-4}, 10^{-5}$. 准确解在 $t = 0$ 处的值是 $y(0) = 1$. 即使是最好的近似, 从初始条件出发, 用了 100 万步到达 $t = 0$, 也是不准确的.

这个例子表明, 为了达到合理的数值计算的精度, 我们需要更精确的方法. 本章其余部分将建立更高级的方法, 它们只要较小的步数就能得到同样的或更好的精度. ◀

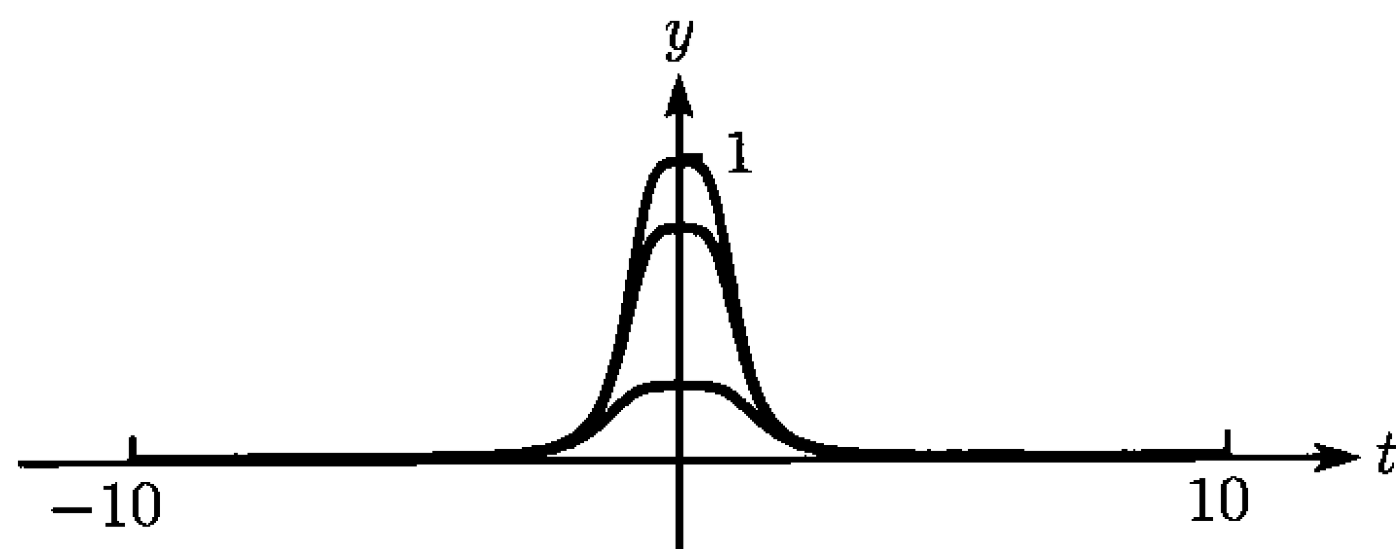


图 6-6 例 6.9 用 Euler 方法得到的近似解. 从底到顶, 步长是 $h = 10^{-3}, h = 10^{-4}, 10^{-5}$. 准确解有 $y(0) = 1$. 为了得到一个合理的近似解, 需要用极其小的步长

6.2.2 显式梯形方法

在 Euler 方法的公式中作一个小的调整便可显著地改进精度. 考虑以下受几何启发的方法:

显式梯形方法

$$\begin{aligned} w_0 &= y_0, \\ w_{i+1} &= w_i + \frac{h}{2}\left(f(t_i, w_i) + f(t_i + h, w_i + hf(t_i, w_i))\right). \end{aligned} \tag{6.29}$$

对 Euler 方法而言, 控制离散步的斜率 $y'(t_i)$ 取自区间 $[t_i, t_{i+1}]$ 的左端点方向场. 而对梯形方法而言, 如图 6-7 所示, 这个斜率被用在左端点的贡献 $y'(t_i)$ 和 Euler 方法已给出的右端点的斜率 $f(t_i + h, w_i + hf(t_i, w_i))$ 的平均来代替. Euler 方法 "预测" 被用于 w 值来计算在 $t_{i+1} = t_i + h$ 处的斜率函数 f. 在某种意义上,Euler 方法预测是用梯形方法来校正的, 正如我们将说明的, 梯形方法更为精确.

梯形方法称为显式方法, 是因为新的近似解 w_{i+1} 能够由前面的 w_i, t_i, h 的显式公式所确定. Euler 方法也是一种显式方法.

名为 "梯形方法" 的原因在于 $f(t, y)$ 不依赖于 y 的特殊情形, 方法

$$w_{i+1} = w_i + \frac{h}{2}[f(t_i) + f(t_i + h)]$$

能被看成把积分 $\int_{t_i}^{t_i+h} f(t)\mathrm{d}t$ 的梯形规则近似添加到当前的 w_i 上去. 因为

$$\int_{t_i}^{t_i+h} f(t)\mathrm{d}t = \int_{t_i}^{t_i+h} y'(t)\mathrm{d}t = y(t_i+h) - y(t_i),$$

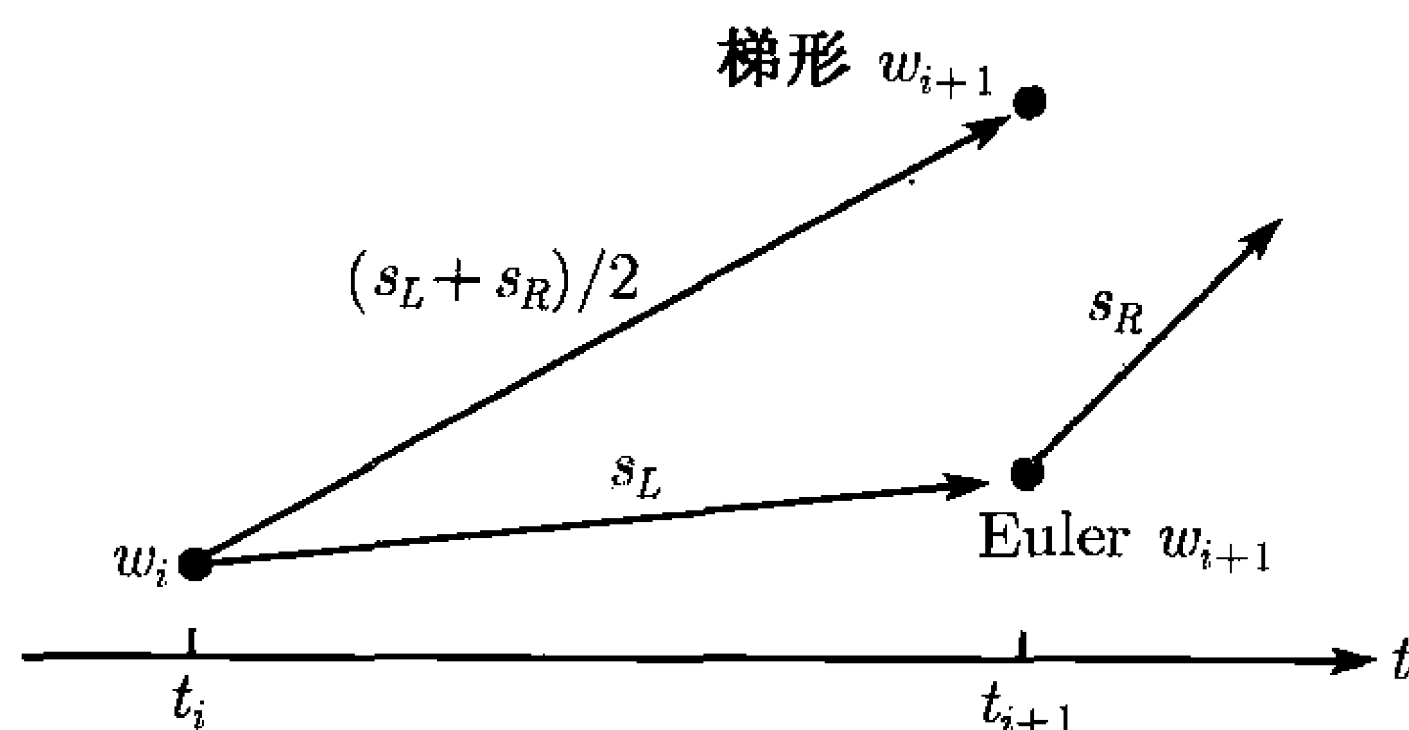


图 6-7　显式梯形方法的单步原理图. 斜率 $s_L = f(t_j, w_j)$ 和 $s_R = f(t_i+h, w_i+hf(t_i, w_i))$ 被平均以定义用于解前进到 t_{i+1} 处的斜率

这对应于用梯形近似积分公式 (5.21) 对微分方程 $y' = f(t)$ 的两边进行积分, 从而求得微分方程的解. 显式梯形方法在文献中也称为改进的 Euler 方法和 Heun 方法, 但是我们将用这一个更形象的及更容易记忆的名称.

例 6.10　在初始条件 $y(0) = 1$ 下, 用显式梯形方法求解初值问题 (6.5).

对于 $f(t,y) = ty + t^3$, 公式 (6.29) 是

$$\begin{aligned} w_0 &= y_0 = 1, \\ w_{i+1} &= w_i + \frac{h}{2}(f(t_i, w_i) + f(t_i+h, w_i + hf(t_i, w_i))) \\ &= w_i + \frac{h}{2}(t_i y_i + t_i^3 + (t_i+h)(w_i + h(t_i y_i + t_i^3)) + (t_i+h)^3). \end{aligned}$$

取步长 $h = 0.1$, 以上迭代便产生了表 6-4. ◀

表　6-4

步骤	t_i	w_i	y_i	e_i
0	0.0	1.000 0	1.000 0	0.000 0
1	0.1	1.005 1	1.005 0	0.000 1
2	0.2	1.020 7	1.020 6	0.000 1
3	0.3	1.048 3	1.048 1	0.000 2
4	0.4	1.090 2	1.089 9	0.000 3
5	0.5	1.149 9	1.149 4	0.000 5
6	0.6	1.232 3	1.231 7	0.000 6
7	0.7	1.343 7	1.342 9	0.000 8
8	0.8	1.492 4	1.491 4	0.001 0
9	0.9	1.689 0	1.687 9	0.001 1
10	1.0	1.947 1	1.946 2	0.001 0

把例 6.10 和对同一问题用 Euler 方法的例 6.2 的结果进行比较, 它们的差别

是惊人的. 为了量化用梯形方法求解初值问题得到的改进, 我们需要计算它的局部截断误差 (6.19).

局部截断误差是在一个单步所产生的误差. 从假定准确解点 (t_i, y_i) 出发, 解在 t_{i+1} 处的正确外延能由 Taylor 展式给出:

$$y_{i+1} = y(t_i + h) = y_i + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{6}y'''(c), \tag{6.30}$$

它对位于 t_i 和 t_{i+1} 之间的常数 c 成立, 其中假定 y''' 是连续的. 为了把这些项与梯形方法进行比较, 将它们写得稍为不同. 对微分方程 $y'(t) = f(t, y)$ 的两边关于 t 微分, 由链式法则得到

$$y''(t) = \frac{\partial f}{\partial t}(t, y) + \frac{\partial f}{\partial y}(t, y)y'(t) = \frac{\partial f}{\partial t}(t, y) + \frac{\partial f}{\partial y}(t, y)f(t, y).$$

(6.30) 的新形式是

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t}(t_i, y_i) + \frac{\partial f}{\partial y}(t_i, y_i)f(t_i, y_i)\right) + \frac{h^3}{6}y'''(c). \tag{6.31}$$

我们想把这个表达式与显式梯形方法进行比较, 利用二维 Taylor 定理来展开下面的项:

$$f(t_i + h, y_i + hf(t_i, y_i)) = f(t_i, y_i) + h\frac{\partial f}{\partial t}(t_i, y_i) + hf(t_i, y_i)\frac{\partial f}{\partial y}(t_i, y_i) + O(h^2).$$

梯形方法能写成

$$\begin{aligned} w_{i+1} &= y_i + \frac{h}{2}(f(t_i, y_i) + f(t_i + h, y_i + hf(t_i, y_i))) \\ &= y_i + \frac{h}{2}f(t_i, y_i) + \frac{h}{2}\left(f(t_i, y_i) + h\left(\frac{\partial f}{\partial t}(t_i, y_i) + f(t_i, y_i)\frac{\partial f}{\partial y}(t_i, y_i)\right) + O(h^2)\right) \\ &= y_i + hf(t_i, y_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t}(t_i, y_i) + f(t_i, y_i)\frac{\partial f}{\partial y}(t_i, y_i)\right) + O(h^3). \end{aligned} \tag{6.32}$$

从 (6.31) 减去 (6.32) 给出局部截断误差为

$$y_{i+1} - w_{i+1} = O(h^3).$$

定理 6.4 说明梯形方法的整体误差正比于 h^2, 这意味着与一阶的 Euler 方法相比, 这个方法是二阶的. 对于小的 h, 这个差别很明显, 回到例 6.9 即可说明这一点.

亮点　复杂性

二阶方法比一阶方法更有效, 亦或相反? 在每一步误差是小的, 但是

计算工作量比较大, 因为通常需要计算两个函数 (与 $f(t,y)$ 有关的) 的值而不是一个. 粗略的比较是这样的: 假设一个近似解在步长 h 时已得到, 我们想使计算量加倍而改进这个近似解. 对于相同数目的函数的求值, 我们 (a) 使一阶方法的步长减半, 用 $\frac{1}{2}$ 乘以整体误差, 或者 (b) 保持相同的步长, 但是用二阶方法, 在定理 6.4 中用 h^2 代替 h, 本质上是用 h 乘以整体误差. 对于小的 h, (b) 较成功.

例 6.11 把梯形方法用于例 6.9:

$$\begin{cases} y' = -4t^3y^2, \\ y(-10) = \dfrac{1}{10\,001}, \\ t \in [-10, 0]. \end{cases}$$

用一个更强的方法重做例 6.9, 在求近似解时得到极大的改进, 例如在 $x = 0$. 取步长 $h = 10^{-3}$, 用梯形方法达到准确值 $y(0) = 1$ 的误差小于 0.015, 恰如图 6-8 所示. 这已给比取步长 $h = 10^{-5}$ 的 Euler 方法更好. 对这个相对困难的初值问题, 取 $h = 10^{-5}$, 梯形方法产生的误差是 $O(10^{-7})$ 阶. ◀

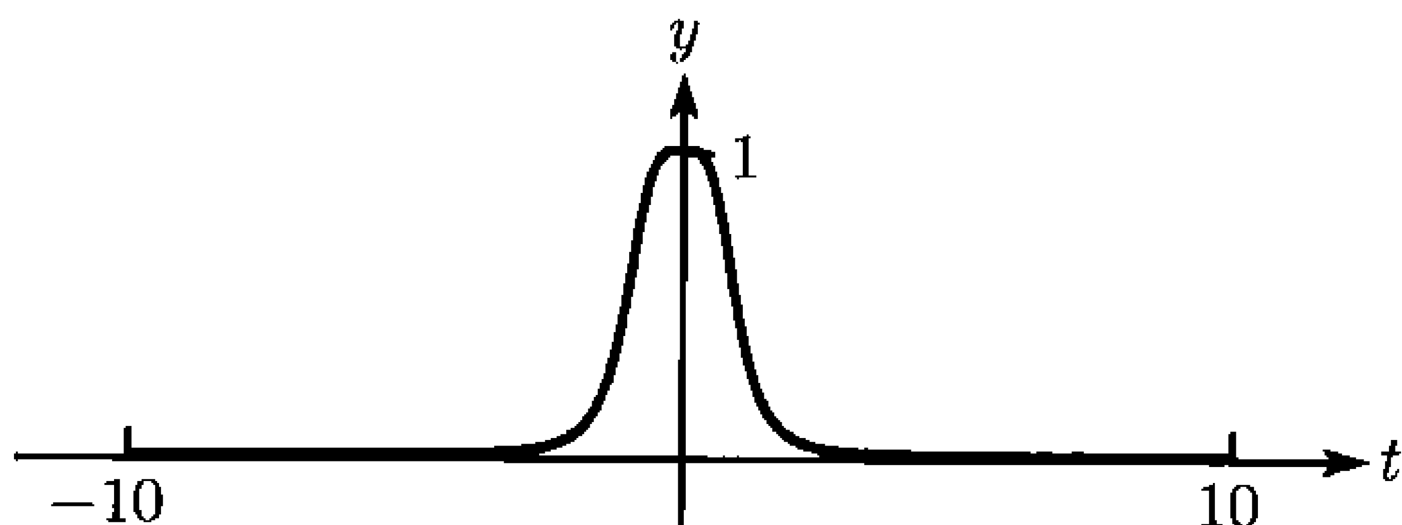


图 6-8 用梯形方法求例 6.9 的近似解: 步长 $h = 10^{-3}$, 注意与图 6-6 中的 Euler 方法相比精度有了重大改进

6.2.3 Taylor 方法

到目前为止, 我们已学习了求常微分方程近似解的两种方法. Euler 方法一阶收敛, 而明显高级的梯形方法二阶收敛. 本节证明存在任意阶的方法. 对每一个正整数 k, 都有一种 k 阶的 Taylor 方法, 这一点我们将在下面叙述.

基本思想是直接利用 Taylor 展开. 假定解 $y(t)$ 是 $(k+1)$ 次连续可微的. 给定解曲线上当前的点 $(t, y(t))$, 目的是对某个步长 h, 利用微分方程的信息, 用 $y(t)$ 来表出 $y(t+h)$. $y(t)$ 关于 t 的 Taylor 展开式是

$$y(t+h) = y(t) + hy'(t) + \frac{1}{2}h^2y''(t) + \cdots + \frac{1}{k!}h^ky^{(k)}(t) + \frac{1}{(k+1)!}h^{k+1}y^{(k+1)}(c), \quad (6.33)$$

其中 c 在 t 和 $t+h$ 之间. 最后一项是 Taylor 余项. 这个等式引发了以下方法:

k 阶 Taylor 方法

$$\begin{aligned} w_0 &= y_0, \\ w_{i+1} &= w_i + hf(t_i, w_i) + \frac{h^2}{2} f'(t_i, w_i) + \cdots + \frac{h^k}{k!} f^{(k-1)}(t_i, w_i). \end{aligned} \tag{6.34}$$

基本记号涉及 $f(t, y(t))$ 关于 t 的全导数. 例如

$$f'(t, y) = f_t(t, y) + f_y(t, y)y'(t) = f_t(t, y) + f_y(t, y)f(t, y).$$

用记号 f_t 表示 f 关于 t 的偏导数, f_y 类似. 为了求 Taylor 方法的局部截断误差, 在 (6.34) 中令 $w_i = y_i$, 并与 Taylor 展式 (6.33) 比较得

$$y_{i+1} - w_{i+1} = \frac{h^{k+1}}{(k+1)!} y^{(k+1)}(c).$$

由此可知: k 阶 Taylor 方法的局部截断误差为 $O(h^{k+1})$, 因此根据定理 6.4, 方法是 k 阶的.

一阶 Taylor 方法是

$$w_{i+1} = w_i + hf(t_i, w_i),$$

它与 Euler 方法相同. 二阶 Taylor 方法是

$$w_{i+1} = w_i + hf(t_i, w_i) + \frac{1}{2}h^2(f_t(t_i, w_i) + f_y(t_i, w_i)f(t_i, w_i)).$$

例 6.12 对问题

$$\begin{cases} y' = ty + t^3, \\ y(0) = y_0, \end{cases} \tag{6.35}$$

建立二阶 Taylor 方法.

由于 $f(t, y) = ty + t^3$, 故有

$$f'(t, y) = f_t + f_y f = y + 3t^2 + t(ty + t^3),$$

因此二阶 Taylor 方法给出

$$w_{i+1} = w_i + h(t_i w_i + t_i^3) + \frac{1}{2}h^2(w_i + 3t_i^2 + t_i(t_i w_i + t_i^3)).$$ ◀

尽管二阶 Taylor 方法是一种二阶方法, 但是注意, 使用者需要手工去求偏导数. 把它与其他的二阶方法相比, (6.29) 仅需要调用计算 $f(t, y)$ 值的程序.

从概念上说,Taylor 方法的经验说明: 存在任意阶的常微分方程方法, 如 (6.34) 所示的那样. 但是我们需要额外计算公式中 f 的所有偏导数. 因为可以建立不需要这些偏导数的相同阶的公式, 所以 Taylor 方法仅仅为了特殊的目的才使用.

习题 6.2

1. 用初始条件 $y(0)=1$ 和步长 $h=\frac{1}{4}$, 在区间 $[0,1]$ 上计算梯形方法近似解 $w_0,\cdots,w_4$. 通过与习题 6.1.3 中求得的正确解比较, 求在 $t=1$ 处的误差.
   (a) $y'=t$;　(b) $y'=t^2y$;　(c) $y'=2(t+1)y$;
   (b) $y'=5t^4y$;　(e) $y'=\dfrac{1}{y^2}$;　(f) $y'=\dfrac{t^3}{y^2}$;
2. 用初始条件 $y(0)=0$ 及步长 $h=\frac{1}{4}$, 在区间 $[0,1]$ 内计算梯形方法近似解. 通过与习题 6.1.4 中求得的正确解比较, 求在 $t=1$ 处的误差.
   (a) $y'=t+y$;　(b) $y'=t-y$;　(c) $y'=4t-2y$.
3. 对下述微分方程, 求二阶 Taylor 方法的公式.
   (a) $y'=ty$;　(b) $y'=ty^2+y^3$;　(c) $y'=y\sin y$;　(d) $y'=e^{yt^2}$.
4. 对习题 1 的初值问题应用二阶 Taylor 方法. 取步长 $h=\frac{1}{4}$, 在区间 $[0,1]$ 上计算二阶 Taylor 方法的近似解. 与习题 6.1.3 中求得的准确解比较, 从而求出在 $t=1$ 处的误差.
5. (a) 证明 (6.22); (b) 证明 (6.23).

计算机问题 6.2

1. 在步长 $h=0.1$ 的网格上, 在 $[0,1]$ 内对习题 1 中的初值问题应用显式梯形方法. 打印每一步的 t 值、近似解及整体截断误差的表.
2. 画出习题 1 中在 $[0,1]$ 上的初值问题的近似解和真解, 步长 $h=0.1,0.05,0.025$.
3. 对于习题 1 中的初值问题, 把在 $t=1$ 处的显式梯形方法的整体截断误差画作 $h=0.1\times 2^{-k}(0\leqslant k\leqslant 5)$ 的函数. 如同图 6-4, 用双对数坐标图.
4. 对于习题 1 中的初值问题, 把在 $t=1$ 处的二阶 Taylor 方法的整体截断误差画作 $h=0.1\times 2^{-k}(0\leqslant k\leqslant 5)$ 的函数.

6.3　常微分方程组

微分方程组的近似求解可以作为单个微分方程的求解方法的简单推广. 解微分方程组大大增强了我们模拟动态行为的能力.

求解常微分方程组的能力是计算机模拟的艺术和科学的核心. 本节介绍两个物理系统, 它们的模拟引发了常微方程解法的极大发展: 摆和轨道力学. 这两个例子的研究将为读者提供某些关于解法的能力和局限性的实际经验.

微分方程的**阶**(order) 指的是方程中出现的最高阶导数. 一阶方程组有形式

$$\begin{cases} y_1'=f_1(t,y_1,\cdots,y_n), \\ y_2'=f_2(t,y_1,\cdots,y_n), \\ \quad\vdots \\ y_n'=f_n(t,y_1,\cdots,y_n). \end{cases}$$

在初值问题中, 每个变量 y_i 需要各自的初始条件.

例 6.13 把 Euler 方法用于两个方程的一阶方程组

$$\begin{cases} y_1' = y_2^2 - 2y_1, \\ y_2' = y_1 - y_2 - ty_2^2, \\ y_1(0) = 0, \\ y_2(0) = 1. \end{cases} \tag{6.36}$$

验证方程组 (6.36) 的解是向量值函数

$$\begin{cases} y_1(t) = t\mathrm{e}^{-2t}, \\ y_2(t) = \mathrm{e}^{-t}. \end{cases}$$

现在假设我们不知道这个解, 而使用 Euler 方法. 把标量 Euler 方法的公式应用到每一个分量:

$$\begin{cases} w_{i+1,1} = w_{i,1} + h(w_{i,2}^2 - 2w_{i,1}), \\ w_{i+1,2} = w_{i,2} + h(w_{i,1} - w_{i,2} - t_i w_{i,2}^2). \end{cases}$$

图 6-9 表示 y_1 和 y_2 的 Euler 方法近似解以及正确解. 执行这一任务的 MATLAB 代码本质上与程序 6.1 相同, 只须稍加调整把 y 处理成向量.

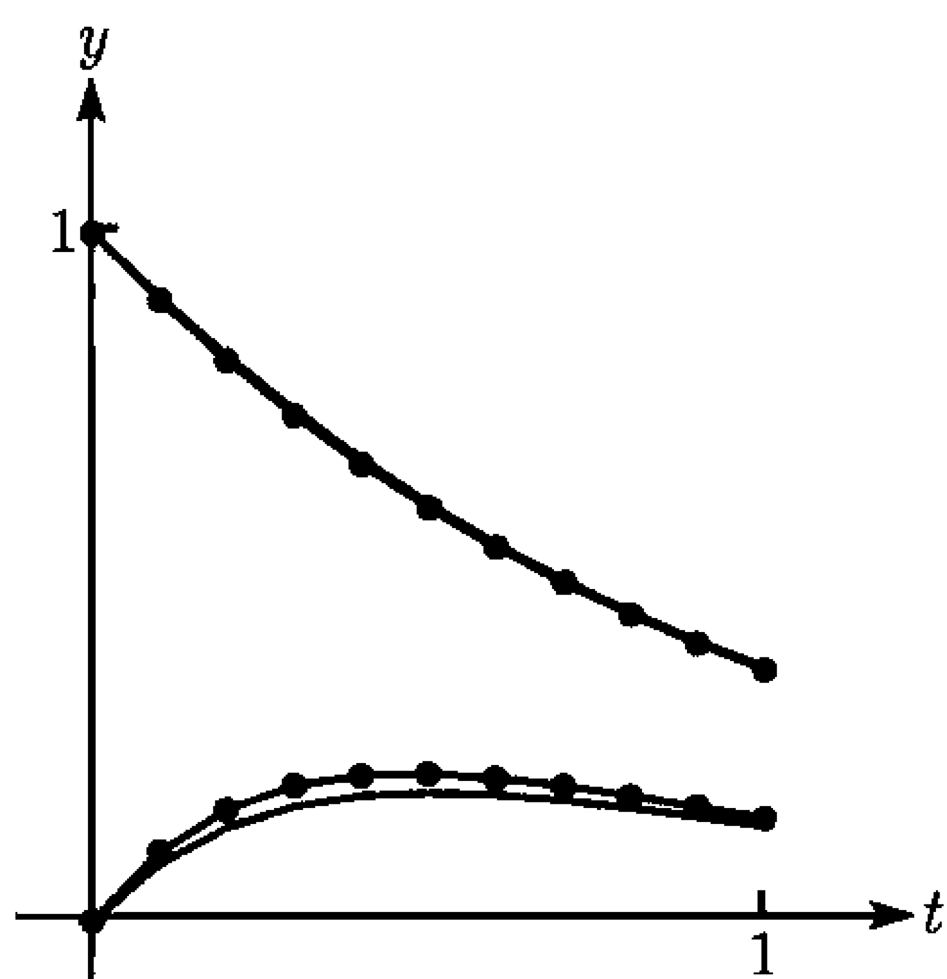


图 6-9 用 Euler 方法近似求解方程 (6.36). 步长 $h = 0.1$, 上面的曲线是 $y_1(t)$ 以及它的近似解 $w_{i,1}$(圆圈者), 而下面的曲线是 $y_2(t)$ 和 $w_{i,2}$

```
% Program 6.2  Vector version of Euler Method
% Input: interval [a,b], initial vector y0, step size h
% Output: time steps t, solution y
% Example usage: euler2([0 1],[0 1],0.1);
function [t,y]=euler2(int,y0,h)
t(1)=int(1); y(1,:)=y0;
n=round((int(2)-int(1))/h);
for i=1:n
```

```
  t(i+1)=t(i)+h;
  y(i+1,:)=eulerstep(t(i),y(i,:),h);
end
plot(t,y(:,1),t,y(:,2));

function y=eulerstep(t,y,h)
%one step of the Euler Method
%Input: current time t, current vector y, step size h
%Output: the approximate solution vector at time t+h
y=y+h*ydot(t,y);

function z=ydot(t,y)
z(1)=y(2)^2-2*y(1);
z(2)=y(1)-y(2)-t*y(2)^2;
```

◀

6.3.1 高阶方程

一个高阶微分方程能够转换成方程组. 令

$$y^{(n)} = f(t, y, y', y'', \cdots, y^{(n-1)})$$

是一个 n 阶常微分方程. 定义新变量

$$y_1 = y, \quad y_2 = y', \quad y_3 = y'', \quad \cdots, \quad y_n = y^{(n-1)},$$

注意到能把原来的微分方程写为

$$y_n' = f(t, y_1, y_2, \cdots, y_n),$$

以及方程

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ y_3' = y_4 \\ \quad \vdots \\ y_{n-1}' = y_n. \end{cases}$$

这样 n 阶微分方程就转换成了一阶方程组, 可以用如 Euler 方法或者梯形方法之类的方法来求解它.

例 6.14 把三阶微分方程

$$y''' = a(y'')^2 - y' + yy'' + \sin t \tag{6.37}$$

转换成一个方程组.

令 $y_1 = y$ 并定义新的变量

$$y_2 = y', \quad y_3 = y''.$$

于是, 用一阶导数来表示,(6.37) 等价于

$$\begin{cases} y_1' = y_2, \\ y_2' = y_3, \\ y_3' = ay_3^2 - y_2 + y_1y_3 + \sin t. \end{cases} \tag{6.38}$$

三阶方程 (6.37) 的解 $y(t)$ 可以通过从方程组 (6.38) 中解出 $y_1(t), y_2(t), y_3(t)$ 求得.

◀

因为可以把高阶方程转化为方程组, 所以我们集中研究一阶方程组. 另外请注意, 用同样的方法可以把几个高阶方程的方程组转换成一阶方程组.

6.3.2 计算机模拟：摆

图 6-10 说明摆在重力影响下摆动. 假定摆悬挂在一个可以自由摆动 360 度的坚硬的杆上. 摆关于垂直方向的角用 y 表示, 所以 $y = 0$ 对应于竖直朝下. 因此,y 和 $y + 2\pi$ 可以认为是相同的角.

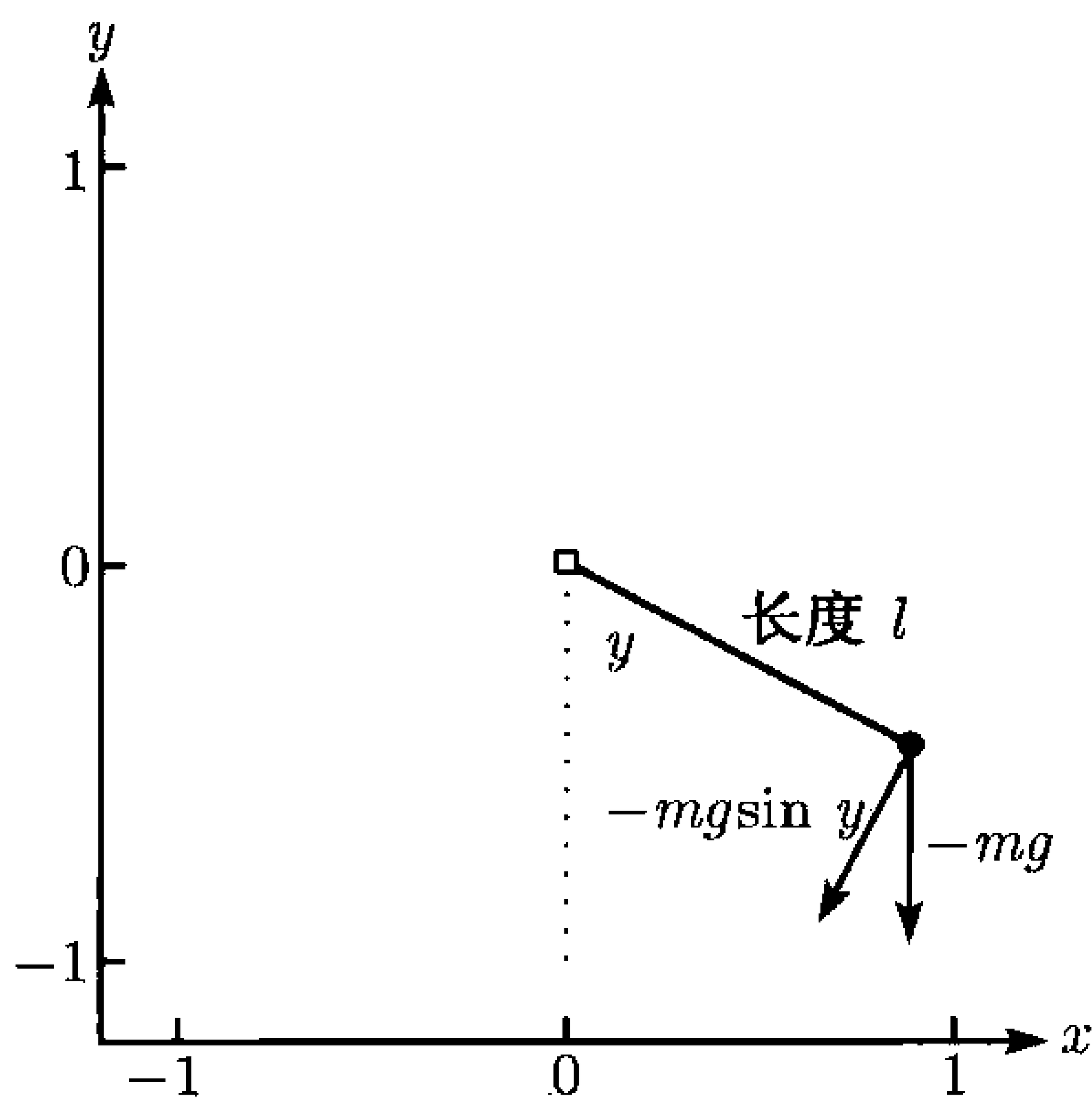


图 6-10 摆：力的切向分量是 $F = -mg\sin y$, 其中 y 是摆在摆动时关于垂直方向的角

可以用 Newton 第二运动定律 $F = ma$ 来求摆的方程. 摆的摆动运动被限制为沿着半径为 l 的圆, 这里 l 是摆杆的长度. 假设 y 以弧度测量, 那么加速度对圆的切向分量是 ly'', 是因为位移对圆的切向分量是 ly. 力沿着运动方向的分量是 mg $\sin y$. 这是一种回复力 (restoring force), 即它的方向与变量 y 的位移方向相反. 因此, 控制光滑摆的微分方程是

$$mly'' = F = -mg\sin y. \tag{6.39}$$

这是摆的角度 y 的二阶微分方程. 初始条件由初始角度 $y(0)$ 及初始角速度 $y'(0)$ 给出.

令 $y_1 = y$ 并引入新的变量 $y_2 = y'$, 这个二阶方程就换成一阶方程组:

$$\begin{cases} y_1' = y_2, \\ y_2' = -\dfrac{g}{l}\sin y_1. \end{cases} \tag{6.40}$$

这个方程组是自控的, 因为其右端与 t 无关. 如果摆从正右方的位置开始, 那么初始条件是 $y_1(0) = \frac{\pi}{2}$ 和 $y_2(0) = 0$, 使用 MKS 单位, 地球表面的重力加速度大概是 9.81 米/秒 ^{2}. 利用这些参数, 我们可以检验 Euler 方法作为这个方程组的解法的适应性.

图 6-11 表示摆方程用两种不同步长的 Euler 方法的近似解. 摆杆长度指定为 $l = 1$ 米. 较小的曲线代表作为时间函数的角度 y. 较大振幅的曲线是瞬时角速度. 注意零角度表示摆的垂直位置, 对应于最大角速度 (正的或负的). 当摆经过最低点时, 摆动最快. 当摆伸展到最右边, 即较小曲线的顶峰时, 速度为零, 它从正向到负向.

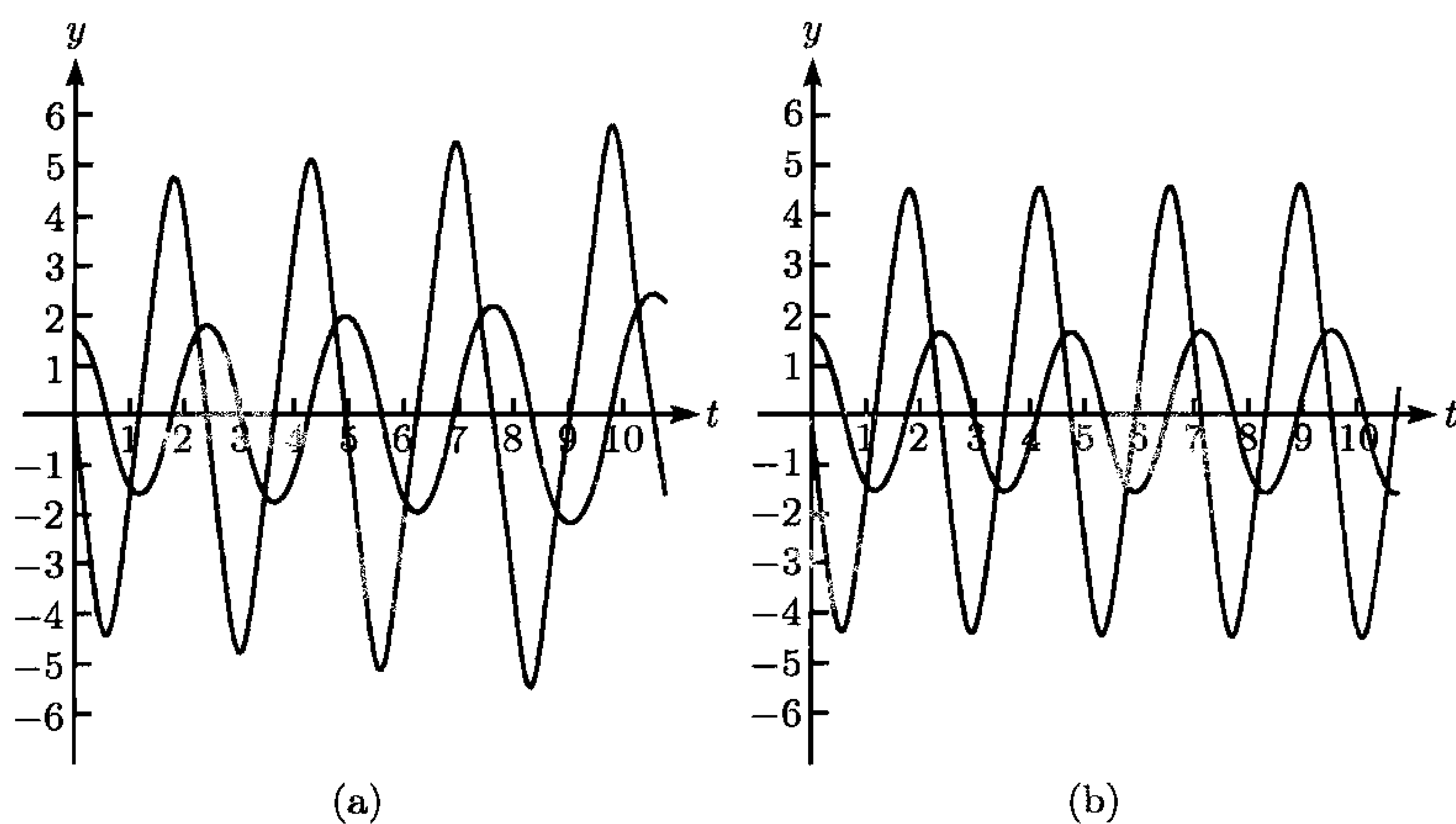


图 6-11 Euler 方法用于摆的方程 (6.40). 振幅较小的曲线是弧度制下的角度 y_1, 振幅较大的曲线是角速度 y_2. (a) 步长 h=0.01 太大了, 能量增长; (b) 步长 $h = 0.001$ 表示更精确的轨道

在图 6-11 中,Euler 方法的缺点是明显的. 为了达到即使是定性的准确性, 步长 $h = 0.01$ 也显然太大了. 以零速度出发的无阻尼摆将永远来回摆动, 以固定的周期回到它出发的位置. 图 6-11a 中角度的幅度增大, 违背了能量守恒. 用 10 倍以上的步数, 如在图 6-11b 中那样, 至少在视觉上改进了这一情形, 但是一共需要 10^4 步, 这对摆所表现的常规动态行为来说, 是一个极大的数字.

像梯形方法这样的二阶常微分方程解法很容易提高了精度. 我们将重写 MATLAB 代码以使用梯形方法, 并且借这一机会来说明 MATLAB 进行简单动画的能力.

下面的代码**pend.m** 包含相同的微分方程信息, 但是用**trapstep**代替了**eulerstep**. 此外还引入了变量**rod**和**bob**来分别表示杆和摆的摆动. MATLAB的**set**命令对变量指定特征属性.**drawnow**命令画出变量**rud**和**bob**. 注意两个变量的删除方法被置为**xor**, 意即当所画变量在某处重新画出时, 前面的位置已被删除. 图 6-10 是这个动画的屏幕截图, 以下便是程序.

```
% Program 6.3 Animation program for pendulum
% Inputs: int=[a b] time interval,
%  initial values ic=[y(1,1) y(1,2)], h = step size
% Calls a one-step method such as trapstep.m
% Example usage: pend([0 10],[pi/2 0],.05)
function pend(int,ic,h)
n=round((int(2)-int(1))/h); % plot n points in total
y(1,:)=ic;                  % enter initial conds in y
t(1)=int(1);
set(gca,'xlim',[-1.2 1.2],'ylim',[-1.2 1.2], ...
  'XTick',[-1 0 1],'YTick',[-1 0 1], ...
  'Drawmode','fast','Visible','on','NextPlot','add');
clf;                       % clear screen
plot(0,0,'ks')             % pivot where rod attached
axis ([xlim,ylim])
axis square                % make aspect ratio 1 - 1
bob=line('color','r','Marker','.','markersize',40,...
    'erase','xor','xdata',[],'ydata',[]);
rod=line('color','b','LineStyle','-','LineWidth',3,...
    'erase','xor','xdata',[],'ydata',[]);
for k=1:n
  t(k+1)=t(k)+h;
  y(k+1,:)=trapstep(t(k),y(k,:),h);
  xbob=cos(y(k+1,1)-pi/2); ybob=sin(y(k+1,1)-pi/2);
  xrod=[0 xbob]; yrod=[0 ybob];
  set(rod,'xdata',xrod,'ydata',yrod)
  set(bob,'xdata',xbob,'ydata',ybob)
  drawnow; pause(h)
end

function y=trapstep(t,x,h)
%one step of the Trapezoid Method
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;
```

```
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;

function z=ydot(t,y)
g=9.81;length=1;
z(1)=y(2);
z(2)=-(g/length)*sin(y(1));
```

在摆方程中应用梯形方法可用较大的步长求得很精确的解. 本节最后给出几个有趣的基本摆模拟的变形, 我们鼓励读者用计算机问题进行实验.

例 6.15　阻尼摆

像空气阻力或者摩擦力这种阻尼力, 经常被模拟为与速度成比例并与速度方向相反, 这种摆的方程组变成

$$\begin{cases} y_1' = y_2, \\ y_2' = -\dfrac{g}{l}\sin y_1 - dy_2, \end{cases} \tag{6.41}$$

这里 $d > 0$ 是阻尼系数. 与无阻尼摆不同, 这种摆随着摆动逐渐失去能量, 并对任意初始条件最终都趋于极限平衡解 $y_1 = y_2 = 0$. 计算机问题 6.3 要求运行阻尼形式的`pend.m`. ◀

例 6.16　受力阻尼摆

在 (6.41) 中加一项依赖于时间的项表示外力作用于阻尼摆. 考虑在 y_2' 的右端加上正弦项 $A\sin t$ 就得到

$$\begin{cases} y_2' = y_2, \\ y_2' = -\dfrac{g}{l}\sin y_1 - dy_2 + A\sin t. \end{cases} \tag{6.42}$$

这可以认为是诸如受到振荡磁场影响的摆的模拟.

在加上外力时, 许多新的动力行为成为可能. 二维自控微分方程组 Poincaré-Bendixson 定理 (来自微分方程理论) 表明, 轨道可以倾向于规则运动, 譬如像摆在低落位置这种稳定的平衡状态, 或者像摆永远前后摆动这种稳定的周期循环状态. 加力使得方程组成为非自控 (它可以写成一个三维自控方程组, 但不能作为一个二维的), 所以允许有第三类轨道, 即混沌轨道. ◀

在计算机问题 4 中, 取阻尼系数 $d = 1$, 外力系数 $A = 10$, 结果得到有趣的周期状态. 变动参数使 $A = 15$, 就引入了混沌轨道.

例 6.17　双摆

双摆是一个简单摆与另一个简单摆一起悬在各自的杆上. 假设两根杆与垂直方向的角度是 y_1 和 y_3, 微分方程组就是

$$\begin{cases} y_1' = y_2, \\ y_2' = \dfrac{-3g\sin y_1 - g\sin(y_1-2y_3) - 2\sin(y_1-y_3)(y_4^2 - y_2^2\cos(y_1-y_3))}{3-\cos(2y_1-2y_3)} - dy_2, \\ y_3' = y_4, \\ y_4' = \dfrac{2\sin(y_1-y_3)[2y_2^2 + 2g\cos y_1 + y_4^2\cos(y_1-y_3)]}{3-\cos(2y_1-2y_3)}. \end{cases}$$

这里 $g = 9.81$, 两根杆子的长度设为 1. 参数 d 表示在摆动中心的摩擦力. 对于 $d = 0$, 双摆展示了对许多初始条件的持续非周期性, 看起来让人着迷, 见计算机问题 8. ◀

6.3.3 计算机模拟: 轨道力学

作为第二个例子, 我们模拟轨道卫星的运动.Newton 第二运动定律讲, 卫星的加速度 a 与作用于卫星上的力 F 有关, $F = ma$, 这里 m 是质量. 根据反平方定律, 重力定律把质量为 m_2 的物体作用到质量为 m_1 的物体的力表达为

$$F = \frac{gm_1m_2}{r^2},$$

这里 r 是两物体之间的距离. 在单体问题中, 其中一个物体的质量与另一个相比认为是可忽略不计, 例如一个小的卫星绕着一个大的行星旋转的情形. 这种简化, 允许我们忽略卫星作用到行星的力, 所以可以认为行星是固定的.

把质量大的物体放在原点, 用 (x, y) 表示卫星的位置. 两物体之间距离 $r = \sqrt{x^2+y^2}$, 因此作用到卫星的力指向中心, 即指向质量大的物体. 方向向量 (这个方向的单位向量) 是

$$\left(-\frac{x}{\sqrt{x^2+y^2}}, -\frac{y}{\sqrt{x^2+y^2}}\right).$$

因此, 用分量表示的作用在卫星上的力是

$$(F_x, F_y) = \left(\frac{gm_1m_2}{x^2+y^2}\frac{-x}{\sqrt{x^2+y^2}}, \frac{gm_1m_2}{x^2+y^2}\frac{-y}{\sqrt{x^2+y^2}}\right). \tag{6.43}$$

把这些力代入牛顿运动定律, 得到两个二阶方程

$$\begin{cases} m_1x'' = -\dfrac{gm_1m_2x}{(x^2+y^2)^{3/2}}, \\ m_1y'' = -\dfrac{gm_1m_2y}{(x^2+y^2)^{3/2}}. \end{cases}$$

引进变量 $v_x = x'$ 及 $y'_y = y'$, 使得这 2 个二阶方程化为 4 个一阶方程的方程组:

$$\begin{cases} x' = v_x, \\ v'_x = -\dfrac{gm_2 x}{(x^2+y^2)^{3/2}}, \\ y' = v_y, \\ v'_y = -\dfrac{gm_2 y}{(x^2+y^2)^{3/2}}. \end{cases} \tag{6.44}$$

以下 MATLAB程序`orbit.m`称为`eulerstep.m`, 它连续地画出卫星轨道.

```
%Program 6.4 Plotting program for one-body problem
%Inputs: int=[a b] time interval, initial conditions
% ic=[x0 vx0 y0 vy0], x position, x velocity, y pos, y vel,
% h=stepsize, p=steps per point plotted
% Calls a one-step method such as trapstep.m
% Example usage: orbit([0 100],[0 1 2 0],0.01,5)
function z=orbit(int,ic,h,p)
n=round((int(2)-int(1))/(p*h));                    % plot n points
x0=ic(1);vx0=ic(2);y0=ic(3);vy0=ic(4);         % grab initial conds
y(1,:)=[x0 vx0 y0 vy0];t(1)=int(1);             % build y vector
set(gca,'XLim',[-5 5],'YLim',[-5 5],'XTick',[-5 0 5],'YTick',...
  [-5 0 5],'Drawmode','fast','Visible','on','NextPlot','add');
cla;
sun=line('color','y','Marker','.','markersize',25,...
  'xdata',0,'ydata',0);
drawnow;
head=line('color','r','Marker','.','markersize',25,...
  'erase','xor','xdata',[],'ydata',[]);
tail=line('color','b','LineStyle','-','erase','none', ...
  'xdata',[],'ydata',[]);
%[px,py,button]=ginput(1);          % include these three lines
%[px1,py1,button]=ginput(1);        % to enable mouse support
%y(1,:)=[px px1-px py py1-py];      % 2 clicks set direction
for k=1:n
  for i=1:p
    t(i+1)=t(i)+h;
    y(i+1,:)=eulerstep(t(i),y(i,:),h);
  end
  y(1,:)=y(p+1,:);t(1)=t(p+1);
  set(head,'xdata',y(1,1),'ydata',y(1,3))
  set(tail,'xdata',y(2:p,1),'ydata',y(2:p,3))
  drawnow;
end

function y=eulerstep(t,x,h)
%one step of the Euler Method
y=x+h*ydot(t,x);
```

```
function y=eulerstep(t,x,h)
%one step of the Euler Method
y=x+h*ydot(t,x);

function z=ydot(t,x)
m2=3;g=1;mg2=m2*g;px2=0;py2=0;
px1=x(1);py1=x(3);vx1=x(2);vy1=x(4);
dist=sqrt((px2-px1)^2+(py2-py1)^2);
z=zeros(1,4);
z(1)=vx1;
z(2)=(mg2*(px2-px1))/(dist^3);
z(3)=vy1;
z(4)=(mg2*(py2-py1))/(dist^3);
```

运行 MATLAB脚本`orbit.m`立刻显示出近似求解有趣问题的 Euler 方法的局限性. 图 6-12a 表示运行`orbit([0 100],[0 1 2 0], 0.01)` 的结果. 换言之, 我们沿着时间区间 $[a,b]=[0,100]$ 上的轨道, 初始位置是 $(x_0,y_0)=(0,2)$, 初始速度是 $(v_x,v_y)=(1,0)$, Euler 步长是 $h=0.01$.

单体问题的解一定是圆锥曲线——椭圆、抛物线或双曲线. 从图 6-12a 中看到的螺旋形是数值人工制品, 意即由计算误差引起的错误代表. 此时 Euler 方法的截断误差导致轨道不能贴近一个椭圆. 如果步长缩小到原来的 $\frac{1}{10}$ 到 $h=0.001$, 结果就得到改进, 如图 6-12b 所示. 所示的几条轨道显然表明, 即使极大地减小步长, 积累的误差依然清晰可见.

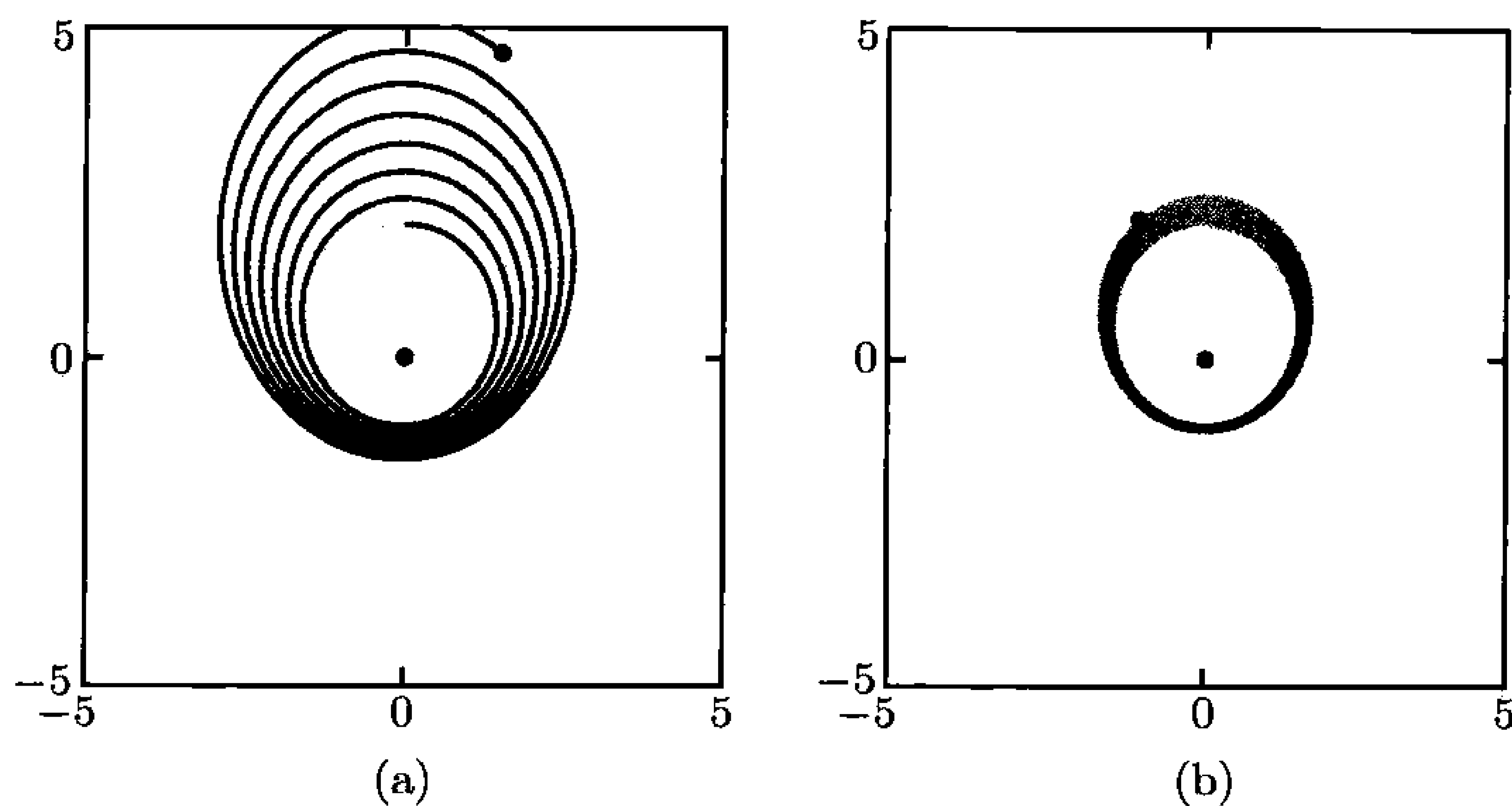


图 6-12 Euler 方法用于单体问题: (a) $h=0.01$,(b) $h=0.001$

推论 6.5 表明, 原则上, 如果步长充分小, Euler 方法逼近解的精度可以如人所愿. 但是图 6-6 和图 6-12 所代表的结果表明, Euler 方法实际上受到严格的限制.

图 6-13 表明, 在单体问题中, 用梯形方法代替 Euler 方法, 可以明显改进结果. 通过在前面的代码中使用`trapstep`代替函数`eulerstep`, 可以画出此图.

在某种意义上，单体问题忽略了卫星作用到 (非常大的) 行星上的力，所以它是虚构的. 当包含后者时，这是两个物体的运动，称之为双体运动.

三个物体在重力下相互作用的情形称为**三体问题**(three-body problem)，它在科学史上占有重要的位置. 即使当所有运动被限制在一个平面 (**约束**三体问题)，但是长期轨道在本质上也是不可预估的. 1889 年，瑞典和挪威的国王 Oscar 二世悬赏证明太阳系的稳定性. Henri Poincaré得到了这一奖项，他指出，由于所看到的三个互相作用的物体的现象，我们几乎不可能去证明任何这类事情.

不可预测性源自对**初始条件的敏感依赖性** (sensitive dependence on initial conditions)，它表达了这样的事实：初始位置和初始速度的微小不确定性可能会导致以后时间上的巨大偏差. 用我们的术语，可以叙述为微分方程组的解关于输入条件和初始条件是病态的.

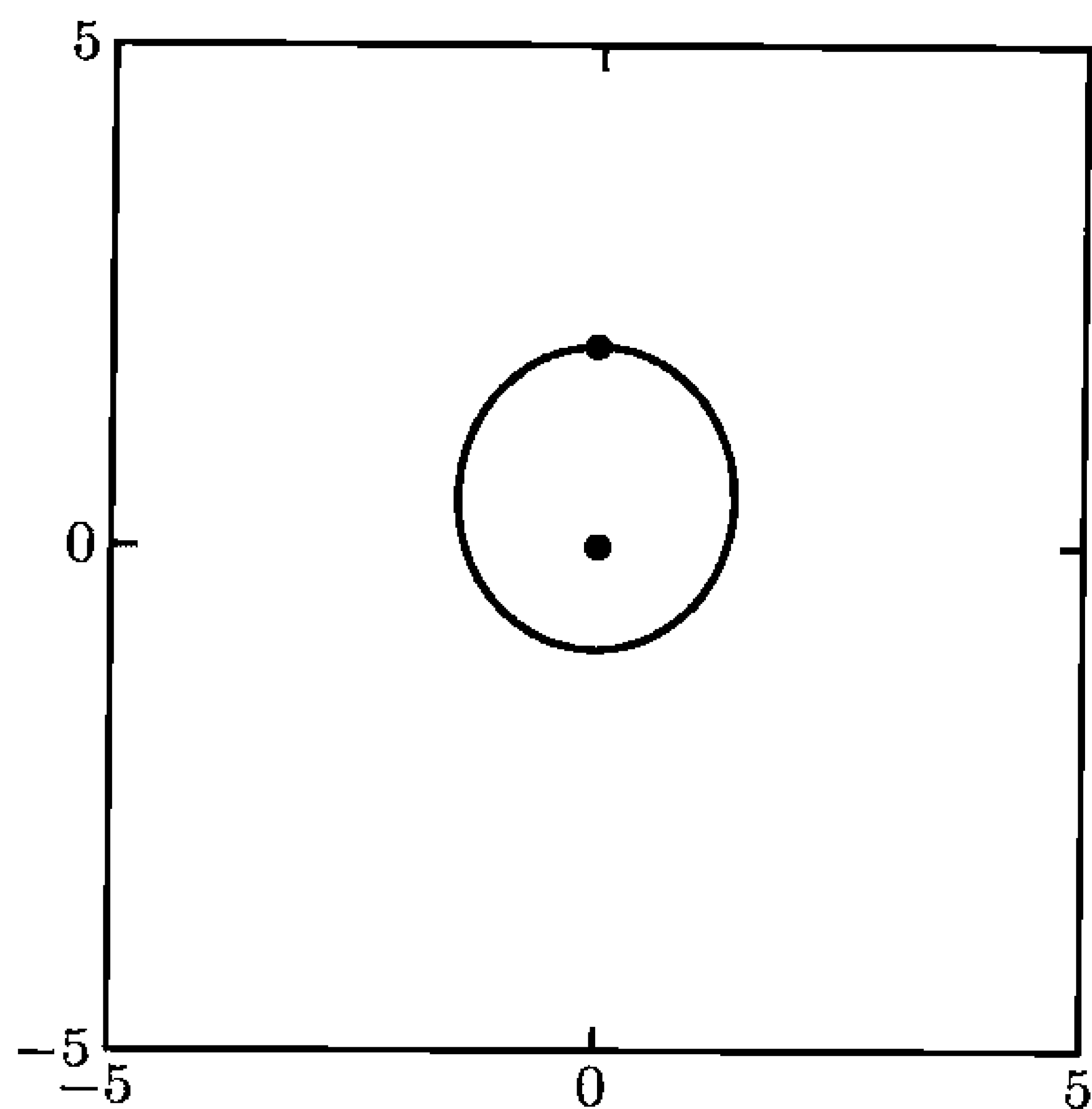


图 6-13 用梯形方法逼近单体问题：步长 $h = 0.01$，至少在图中的可见解处轨道似乎是封闭的

约束三体问题是一个 12 个方程的方程组，每一个物体对应 4 个方程，它们同样由 Newton 第二定律导出. 例如，第一个物体的方程是

$$\begin{cases} x_1' = v_{1x}, \\ v_{1x}' = \dfrac{gm_2(x_2 - x_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^{3/2}} + \dfrac{gm_3(x_3 - x_1)}{((x_3 - x_1)^2 + (y_3 - y_1)^2)^{3/2}}, \\ y_1' = v_{1y}, \\ v_{1y}' = \dfrac{gm_2(y_2 - y_1)}{((x_2 - x_1)^2 + (y_2 - y_1)^2)^{3/2}} + \dfrac{gm_3(y_3 - y_1)}{((x_3 - x_1)^2 + (y_3 - y_1)^2)^{3/2}}. \end{cases} \tag{6.45}$$

在 (x_2, y_2) 和 (x_3, y_3) 处的第二个物体和第三个物体分别满足类似的方程.

计算机问题 9 和问题 10 要求读者计算地求解二体问题和三体问题. 后一问题说明了严重的对初始条件的敏感依赖性.

习题 6.3

1. 取步长 $h=\frac{1}{4}$, 用 Euler 方法解 $[0,1]$ 上的初值问题:

(a) $\begin{cases} y_1'=y_1+y_2, \\ y_2'=-y_1+y_2, \\ y_1(0)=1, \\ y_2(0)=0; \end{cases}$ (b) $\begin{cases} y_1'=-y_1-y_2, \\ y_2'=y_1-y_2, \\ y_1(0)=1, \\ y_2(0)=0; \end{cases}$

(c) $\begin{cases} y_1'=-y_2, \\ y_2'=y_1, \\ y_1(0)=1, \\ y_2(0)=0; \end{cases}$ (d) $\begin{cases} y_1'=y_1+3y_2, \\ y_2'=2y_1+2y_2, \\ y_1(0)=5, \\ y_2(0)=0. \end{cases}$

通过与准确解比较, 求 $t=1$ 处的 y_1 和 y_2 的整体截断误差:

(a) $y_1(t)=\mathrm{e}^t\cos t, y_2(t)=-\mathrm{e}^t\sin t$; **(b)** $y_1(t)=\mathrm{e}^{-t}, y_2(t)=\mathrm{e}^{-t}\sin t$;

(c) $y_1(t)=\cos t, y_2(t)=\sin t$; **(d)** $y_1(t)=3\mathrm{e}^{-t}+2\mathrm{e}^{4t}, y_2(t)=-2\mathrm{e}^{-t}+2\mathrm{e}^{4t}$.

2. 取步长 $h=\frac{1}{4}$, 用梯形方法解习题 1 中的初值问题. 通过与准确解比较, 求 $t=1$ 处的整体截断误差.

3. 把以下高阶常微分方程化为一阶方程组.

(a) $y''-ty=0$ (Airy 方程); **(b)** $y''-2ty'+2y=0$(Hermite 方程); **(c)** $y''-ty-y=0$.

4. 取 $h=\frac{1}{4}, y(0)=y'(0)=1$, 用梯形方法解习题 3 中的初值问题.

5. **(a)** 证明 $y(t)=(\mathrm{e}^t+\mathrm{e}^{-t})/2-1$ 是初值问题 $y'''-y'=t, y(0)=y'(0)=y''(0)=0$ 的解. **(b)** 把微分方程化为 3 个一阶方程的方程组. **(c)** 取 $h=\frac{1}{4}$, 用 Euler 方法求 $[0,1]$ 上的近似解. **(d)** 求 $t=1$ 处的整体截断误差.

计算机问题 6.3

1. 取 $h=0.1$ 及 $h=0.01$, 用 Euler 方法解习题 1 中的初值问题. 画出 $[0,1]$ 上的近似解和准确解. 并求 $t=1$ 处的整体截断误差. 取 $h=0.01$ 时, 误差的减小与 Euler 方法的阶是否一致?

2. 用梯形方法执行计算机问题 1.

3. 修改`pend.m`, 对阻尼摆建模. 运行结果码 ($d=0.1$). 除了初始条件 $y_1(0)=\pi, y_2(0)=0$, 随着时间的推移, 所有的轨道都朝着直线向下运动. 验证例外的初始条件: 这种模拟与理论一致吗? 与物理摆呢?

4. 修改`pend.m`, 构造一个受力阻尼摆. 把梯形方法用到: **(a)** 令阻尼 $d=1$ 及受力参数 $A=10$. 取步长 $h=0.005$ 及你选取的初始条件. 在几个暂时不稳定的运动之后, 摆将进入周期 (重复) 轨道. 定性描述这种轨道. 尝试不同的初始条件. 所有的解是否都结束于相同的 "有吸引力的" 周期轨道? **(b)** 现在把步长增大到 $h=0.01$, 重复这一实验. 尝试初始条件 $[\frac{\pi}{2},0]$ 和其他的初始条件. 描述所发生的一切, 并对在这一步长处的反常行为给出合理的解释.

5. 如在计算机问题 4 中那样运行受力阻尼振动, 但令 $A=12$. 取 $h=0.005$, 应用梯形方法. 存在两个互为镜像的周期吸引轨道. 描述这两条吸引轨道, 求被不同周期轨道吸引的

两个初始条件 $(y_1, y_2) = (a, 0)$ 和 $(b, 0)$, 这里 $|a-b| \leqslant 0.1$. 令 $A = 15$, 观察受力阻尼摆的混沌运动.

6. 修改`pend.m`, 以构造一个带有振荡 (摇摆) 轴的阻尼摆. 目的是研究参量共振现象. 由此, 倒置的摆成为稳定的! 方程是

$$y'' + dy' + \left(\frac{g}{l} + A\cos 2\pi t\right)\sin y = 0,$$

这里 A 是外力强度. 令 $d = 0.1$, 摆的长度是 2.5 米. 在没有外力 $(A = 0)$ 时, 向下的摆 $y = 0$ 是一种稳定的平衡状态. 而倒置的摆 $y = \pi$ 是一种不稳定平衡. 求出尽可能精确的参数 A 的范围, 使得倒置摆在其中成为稳定 (当然 $A = 0$ 太小, 而事实表明 $A = 30$ 又太大). 在你的试验中用初始条件 $y = 3.1$; 如果摆不经过向下的位置, 称倒置的摆 "稳定".

7. 应用计算机问题 6 的参数设置来说明参数共振的其他影响: 振荡轴能使稳定平衡变成不稳定. 求出使这种情形发生的受力强度 A 的最小 (正) 值. 如果摆最终进入倒置位置, 把向下的位置归入不稳定一类.

8. 修改`pend.m`, 以构造双摆. 对第二个摆必须定义一对新的杆与摆. 注意第二根杆的末端等于第一根杆的自由端. 用简单的三角学可以计算第二根杆的自由端的位置 (x, y).

9. 修改`orbit.m`, 求解双体问题. 令质量 $m_2 = 0.3, m_1 = 0.03$, 并且取初始条件 $(x_1, y_1) = (2,2), (x_1', y_1') = (0.2, -0.2), (x_2, y_2) = (0,0), (x_2', y_2') = (-0.01, 0.01)$, 画出相应的轨道.

10. 修改`orbit.m`, 求解三体问题. 设质量 $m_2 = 0.3, m_1 = m_3 = 0.03$. (a) 取初始条件 $(x_1, y_1) = (2,2), (x_1', y_1') = (0.2, -0.2), (x_2, y_2) = (0,0), (x_2', y_2') = (0,0), (x_3, y_3) = (-2,-2), (x_3', y_3') = (-0.2, 0.2)$. (b) 把初始条件 x_1' 改为 0.200 01, 并比较结果轨道. 这个例子非常直观地表明了敏感依赖.

6.4 Runge-Kutta 方法及其应用

Runge-Kutta 方法是包括 Euler 方法和梯形方法在内的一族常微分方程解法, 也是更复杂的高阶方法. 本节介绍各种单步方法, 并用它们模拟某些关键应用的轨道.

6.4.1 Runge-Kutta 族

我们已经看到: Euler 方法是一阶方法, 梯形方法是二阶方法. 除了梯形方法外, 还有其他 Runge-Kutta 型的二阶方法. 一个重要的例子是**中点方法**.

中点方法

$$\begin{aligned} w_0 &= y_0, \\ w_{i+1} &= w_i + hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)\right). \end{aligned} \tag{6.46}$$

为了查证中点方法的阶, 必须计算它的局部截断误差. 我们对梯形方法做这个工作时, 发现表达式 (6.31) 是有用的:

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial t}(t_i, y_i) + \frac{\partial f}{\partial y}(t_i, y_i)f(t_i, y_i)\right) + \frac{h^3}{6}y'''(c). \tag{6.47}$$

为了计算在第 i 步的局部截断误差, 我们假定 $w_i = y_i$, 并计算 $y_{i+1} - w_{i+1}$. 像梯形方法那样, 重复利用 Tayler 展开, 可得

$$\begin{aligned} w_{i+1} &= y_i + hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}f(t_i, y_i)\right) \\ &= y_i + h\left(f(t_i, y_i) + \frac{h}{2}\frac{\partial f}{\partial t}(t_i, y_i) + \frac{h}{2}f(t_i, y_i)\frac{\partial f}{\partial y}(t_i, y_i) + O(h^2)\right). \end{aligned} \tag{6.48}$$

比较 (6.47) 和 (6.48) 就得到

$$y_{i+1} - w_{i+1} = O(h^3),$$

所以根据定理 6.4, 中点方法是二阶的.

微分方程右边的每个函数的计算称为方法的**级**. 梯形方法和中点方法都是二阶 Runge-Kutta 方法二级族中的成员, 它们具有形式

$$w_{i+1} = w_i + h\left(1 - \frac{1}{2\alpha}\right)f(t_i, w_i) + \frac{h}{2\alpha}f(t_i + \alpha h, w_i + \alpha hf(t_i, w_i)), \tag{6.49}$$

其中 $\alpha \neq 0$. 令 $\alpha = 1$ 就对应于显式梯形方法, 而 $\alpha = \frac{1}{2}$ 就对应于中点方法. 习题 5 要求你证明这族方法的阶.

亮点　收敛性

像 RK4 这种 4 阶方法的收敛性质大大优于到目前为止我们讨论过的一阶和二阶方法的收敛性质. 这里收敛意味着在某一固定的时间 t, 常微分方程近似的 (整体) 误差随着步长 h 趋于零而趋于零的速度有多快. 4 阶的意思是每次步长减半, 误差近似地下降到 $\left(\frac{1}{2}\right)^4 = \frac{1}{16}$, 这在图 6-15 中很清楚.

图 6-14 说明了梯形方法和中点方法背后的直觉. 梯形方法用一个 Euler 步到达区间的右端点, 计算在这点的斜率, 然后与左端点的斜率进行平均. 中点方法用一个 Euler 步移动到区间的中点, 算出在该点的斜率是 $f\left(t_i + \frac{h}{2}, w_i + \left(\frac{h}{2}\right)f(t_i, w_i)\right)$, 再用这个斜率把 w_i 移动到新的近似 w_{i+1}. 这些方法用不同的逼近来解同样的问题: 要求代表整体区间的斜率比 Euler 方法好, Euler 方法仅仅用了区间左端点的斜率估计.

存在任意阶的 Runge-Kutta 方法. 特别普遍存在的例子是 4 阶方法.

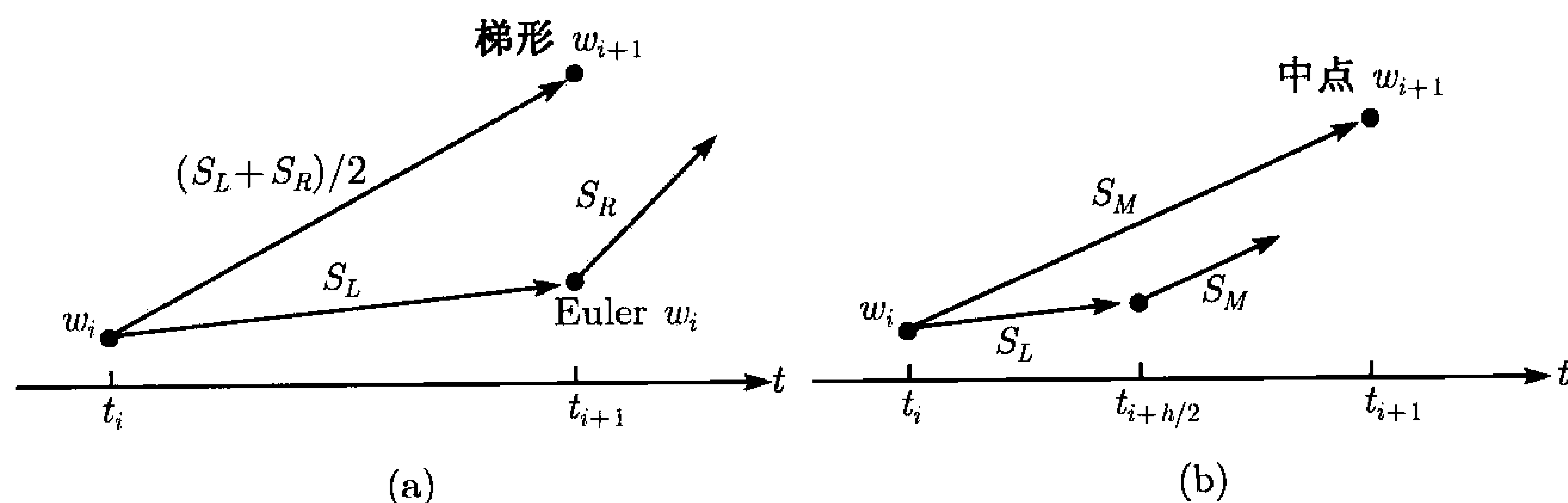


图 6-14 RK2 族中两个成员的示意图: (a) 梯形方法用左右端点的平均值通过区间;(b) 中点方法用区间中点的斜率

4 阶 Runge-Kutta 方法 (RK4)

$$w_{i+1} = w_i + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4), \tag{6.50}$$

其中

$$s_1 = f(t_i, w_i), \qquad s_2 = f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_1\right),$$

$$s_3 = f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_2\right), \qquad s_4 = f(t_i + h, w_i + hs_3).$$

这种方法之所以流行, 是因为它简单和容易编程. 它是单步方法, 所以它只需要一个初始条件就可起动; 还有一种 4 阶方法, 它显然比 Euler 方法或者梯形方法更精确.

在 4 阶 Runge-Kutta 方法中, $h(s_1 + 2s_2 + 2s_3 + s_4)/6$ 代替了 Euler 方法中的斜率. 可以考虑这个量作为在区间 $[t_i, t_i + h]$ 中的解的斜率的一种改进估计. 注意 s_1 是在区间左端点的斜率, s_2 是用于中点方法的斜率, s_3 是在中点处改进后的斜率, 而 s_4 是在右端点 $t_i + h$ 处的近似斜率. 代数上需要证明这个方法是 4 阶的, 类似于梯形方法和中点方法的推导, 但是有点长, 并且可以在譬如 [14] 中找到. 为了进行比较, 我们再次回到微分方程 (6.5).

例 6.18 把 4 阶 Runge-Kutta 方法用于初值问题

$$\begin{cases} y' = ty + t^3, \\ y(0) = 1. \end{cases} \tag{6.51}$$

在 $t = 1$ 处对各种步长计算整体截断误差, 结果如表 6-5.

与 Euler 方法的表 6-3 进行比较. 差别是显著的, 并且容易补偿 RK4 的额外的复杂性. 与 Euler 方法每步仅需调用 1 个函数相比, RK4 每步要调用 4 个函数.

图 6-15 用展示整体截断误差正比于 h^4 这个事实的方式来显示同样的信息. 这就像我们对 4 阶方法所期望的那样. ◀

表 6-5

步骤 n	步长 h	在 $t=1$ 处的误差
5	0.200 00	$2.378\,8\times10^{-5}$
10	0.100 00	$1.465\,5\times10^{-6}$
20	0.050 00	$9.035\,4\times10^{-8}$
40	0.025 00	$5.598\,3\times10^{-9}$
80	0.012 50	$3.482\,0\times10^{-10}$
160	0.006 25	$2.171\,0\times10^{-11}$
320	0.003 12	$1.349\,1\times10^{-12}$
640	0.001 57	$7.260\,9\times10^{-4}$

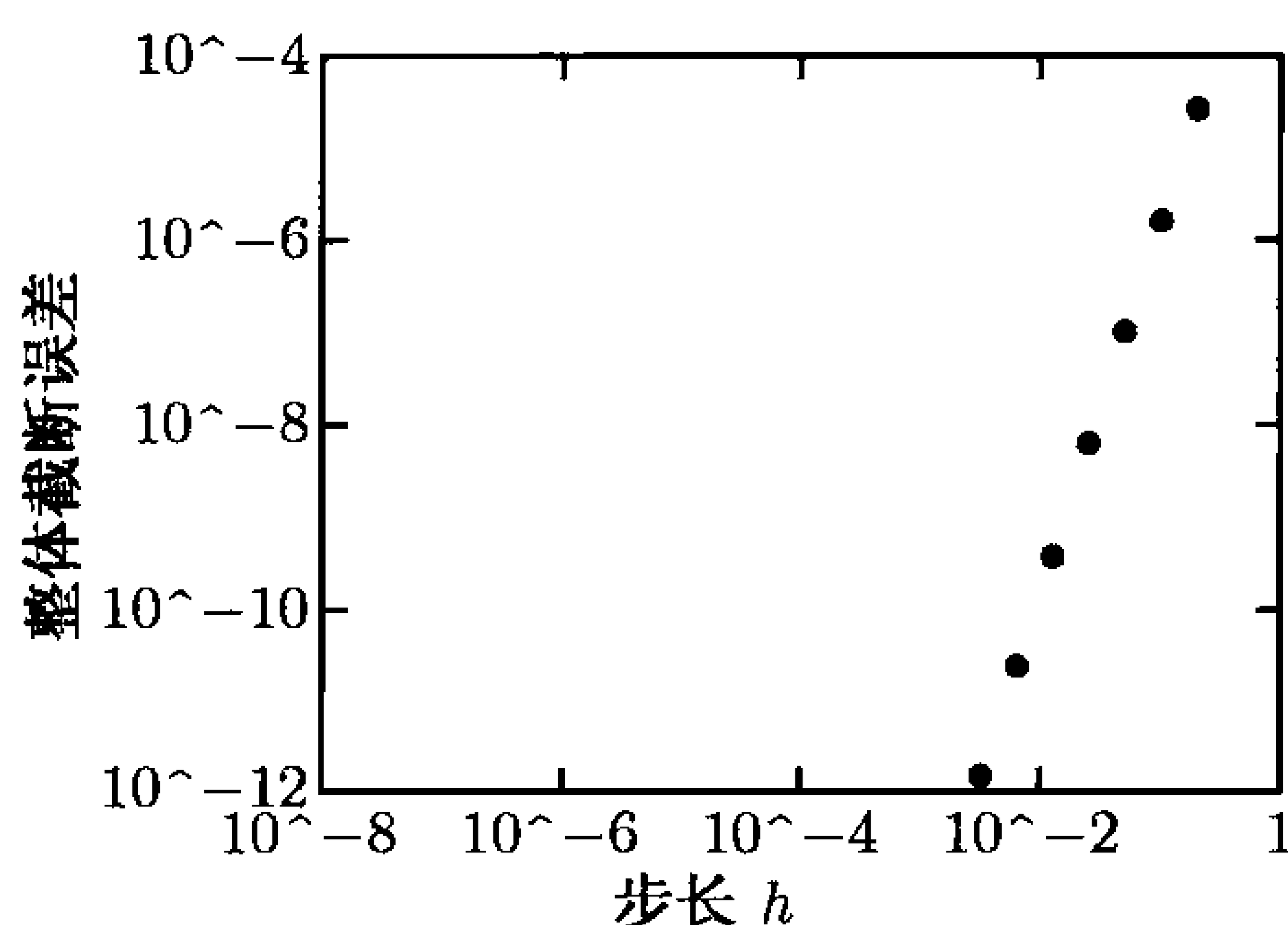


图 6-15　4 阶 Runge-Kutta 方法中作为步长函数的误差: 在双对数坐标图中, 在 $t=1$ 处 (6.5) 的近似解和准确解的差的斜率等于 4, 所以对小的 h, 它正比于 h^4

6.4.2 计算机模拟: Hodgkin-Huxley 神经元

20 世纪中叶, 计算机还在早期发展阶段. 最初的几种应用是帮助求解至今都很难对付的微分方程组.

A. L. Hodgkin 和 A. F. Huxley 通过建立神经细胞或神经元的仿真启动模型, 使计算机神经科学 (computational neuroscience) 得以诞生. 他们甚至能用那个时代的原始的计算机来近似求解这种微分方程模型. 因为这个工作, 他们获得了 1963 年诺贝尔生物学奖.

这个模型是一个 4 个耦合微分方程的方程组, 其中一个模拟了细胞内外的电位差. 其余三个方程模拟离子轨道的激活水准. 它们起了改变内外钠离子和钾离子的作用. **Hodgkin-Huxley 方程是**

$$
\begin{aligned}
Cv' &= -g_1m^3h(v-E_1) - g_2n^4(v-E_2) - g_3(v-E_3) + I_{\text{输入}},\\
m' &= (1-m)\alpha_m(v-E_0) - m\beta_m(v-E_0),
\end{aligned}
$$

$$\begin{aligned}n' &= (1-m)\alpha_n(v-E_0) - n\beta_n(v-E_0),\\ h' &= (1-h)\alpha_h(v-E_0) - h\beta_h(v-E_0),\end{aligned} \tag{6.52}$$

其中

$$\alpha_m(v) = \frac{2.5-0.1v}{e^{2.5-0.1v}-1}, \qquad \beta_m(v) = 4e^{-v/18},$$

$$\alpha_n(v) = \frac{0.1-0.01v}{e^{1-0.1v}-1}, \qquad \beta_n(v) = \frac{1}{8}e^{-v/80},$$

$$\alpha_h(v) = 0.07e^{-\frac{v}{20}}, \qquad \beta_h(v) = \frac{1}{e^{3-0.1v}+1}.$$

系数 C 表示细胞的电容, $I_{输入}$ 表示来自其他细胞的输入电流. 典型的系数值是电容 $C=1$ 毫法拉, 传导系数 $g_1=120, g_2=36, g_3=0.3$, 电压 $E_0=-65, E_1=50, E_2=-77, E_3=-54.4$ 毫伏.

v' 的方程是一个每单位面积电流的方程, 单位为毫安 (培)/厘米 ^{2}, 而其余 3 个激活 (因子)m,n,h 是无单位的. 系数 C 是神经薄膜的电容 ,g_1,g_2,g_3 是传导系数, 而 E_1,E_2,E_3 是 "反势能", 即是形成向内流和向外流的边界的电压水平.

Hudgkin 和 Huxley 仔细地选择方程的类型去匹配经验数据, 这些经验数据来自乌贼的巨轴突. 他们也拟合这个模型中的参数. 虽然乌贼轴突的详细资料不同于哺乳动物的神经, 但是这个模型已作为神经动力学的一种实际描绘. 更普遍地, 把连续输入转化为全有感应或全无感应的激发介质的例子是有用的.

执行这个模型的 MATLAB 代码如下:

```
% Program 6.5 Hodgkin-Huxley equations
% Inputs: [a b] time interval,
% ic=initial voltage v and 3 gating variables, step size h
% Output: solution y
% Calls a one-step method such as rk4step.m
% Example usage: hh([0,100],[-65,0,0.3,0.6],0.05);
function y=hh(inter,ic,h)
global pa pb pulse
inp=input('pulse start, end, muamps in [ ], e.g. [50 51 7]: ');
pa=inp(1);pb=inp(2);pulse=inp(3);
a=inter(1); b=inter(2); n=ceil((b-a)/h); % plot n points in total
y(1,:)=ic;                               % enter initial conds in y
t(1)=a;
for i=1:n
  t(i+1)=t(i)+h;
  y(i+1,:)=rk4step(t(i),y(i,:),h);
end
subplot(3,1,1);
plot([a pa pa pb pb b],[0 0 pulse pulse 0 0]);
grid;axis([0 100 0 2*pulse])
ylabel('input pulse')
subplot(3,1,2);
```

```
plot(t,y(:,1));grid;axis([0 100 -100 100])
ylabel('voltage (mV)')
subplot(3,1,3);
plot(t,y(:,2),t,y(:,3),t,y(:,4));grid;axis([0 100 0 1])
ylabel('gating variables')
legend('m','n','h')
xlabel('time (msec)')

function y=rk4step(t,w,h)
%one step of the Runge-Kutta order 4 method
s1=ydot(t,w);
s2=ydot(t+h/2,w+h*s1/2);
s3=ydot(t+h/2,w+h*s2/2);
s4=ydot(t+h,w+h*s3);
y=w+h*(s1+2*s2+2*s3+s4)/6;

function z=ydot(t,w)
global pa pb pulse
c=1;g1=120;g2=36;g3=0.3;T=(pa+pb)/2;len=pb-pa;
e0=-65;e1=50;e2=-77;e3=-54.4;
in=pulse*(1-sign(abs(t-T)-len/2))/2;
% square pulse input on interval [pa,pb] of pulse muamps
v=w(1);m=w(2);n=w(3);h=w(4);
z=zeros(1,4);
z(1)=(in-g1*m*m*m*h*(v-e1)-g2*n*n*n*n*(v-e2)-g3*(v-e3))/c;
v=v-e0;
z(2)=(1-m)*(2.5-0.1*v)/(exp(2.5-0.1*v)-1)-m*4*exp(-v/18);
z(3)=(1-n)*(0.1-0.01*v)/(exp(1-0.1*v)-1)-n*0.125*exp(-v/80);
z(4)=(1-h)*0.07*exp(-v/20)-h/(exp(3-0.1*v)+1);
```

没有输入, Hodgkin-Huxley 神经元在近似 E_0 的电压处保持静止. 使 $I_{输入}$ 是长度为 1 微秒和强度为 7 微安的方波电流脉冲就足以造成尖峰信号, 即电压的极化修正. 图 6-16 说明了这一点. 运行这个程序来验证 $6.9\mu A$ 不一定造成一个完整的尖峰信号. 因此产生全有感应或全无感应. 正是输入中微小差别的极大放大效应可以解释信息处理中神经元的成功. 图 6-16b 说明, 假如输入电流持续, 那么神经元将周期地齐射尖峰. 计算机问题 6 是对这种虚拟神经元的初期能力的研究.

6.4.3 计算机模拟: Lorenz 方程

在 20 世纪 50 年代后期, 麻省理工学院的气象学家 E. Lorenz 得到了第一台商用计算机. 它有冰箱那么大, 以每秒 60 次乘法的速度运转, 借助于这种惊人的计算能力, 他对由几个微分方程组成的气象模型进行了有意义的计算, 这些微分方程像 Hodgkin-Huxlay 方程那样, 不能用解析方法求解.

Lorenz 方程是一个微型大气模型的简化形式, 他设计这个模型来研究 Rayleih-Bénard 对流, 这是热在流体 (例如空气) 中从较低的热介质 (例如地面) 到较高的

冷介质 (例如上面的大气层) 的运动. 在这个二维大气模型中形成了空气循环, 这可用以下的 3 个方程的方程组来描述:

$$\begin{cases} x' = -sx + sy, \\ y' = -xz + rx - y, \\ z' = xy - bz. \end{cases} \tag{6.53}$$

变量 x 表示顺时针循环的速度, y 表示上升和下降空气柱之间的温度差, 而 z 度量了来自垂直方向的严格线性温度剖面的偏差. Prandtl 数 s、Reynolds 数 r 及 b 是方程组的参数. 在大多数情况下, 取参数为 $s = 10, r = 28, b = \frac{8}{3}$. 参数的这些取法用于如图 6-17 所示的轨道, 它是用以下描述微分方程的代码通过 4 阶 Runge-Kutta 方法计算而得到的.

```
function z=ydot(t,y)
%Lorenz equations
s=10; r=28; b=8/3;
z(1)=-s*y(1)+s*y(2);
z(2)=-y(1)*y(3)+r*y(1)-y(2)
z(3)=y(1)*y(2)-b*y(3)
```

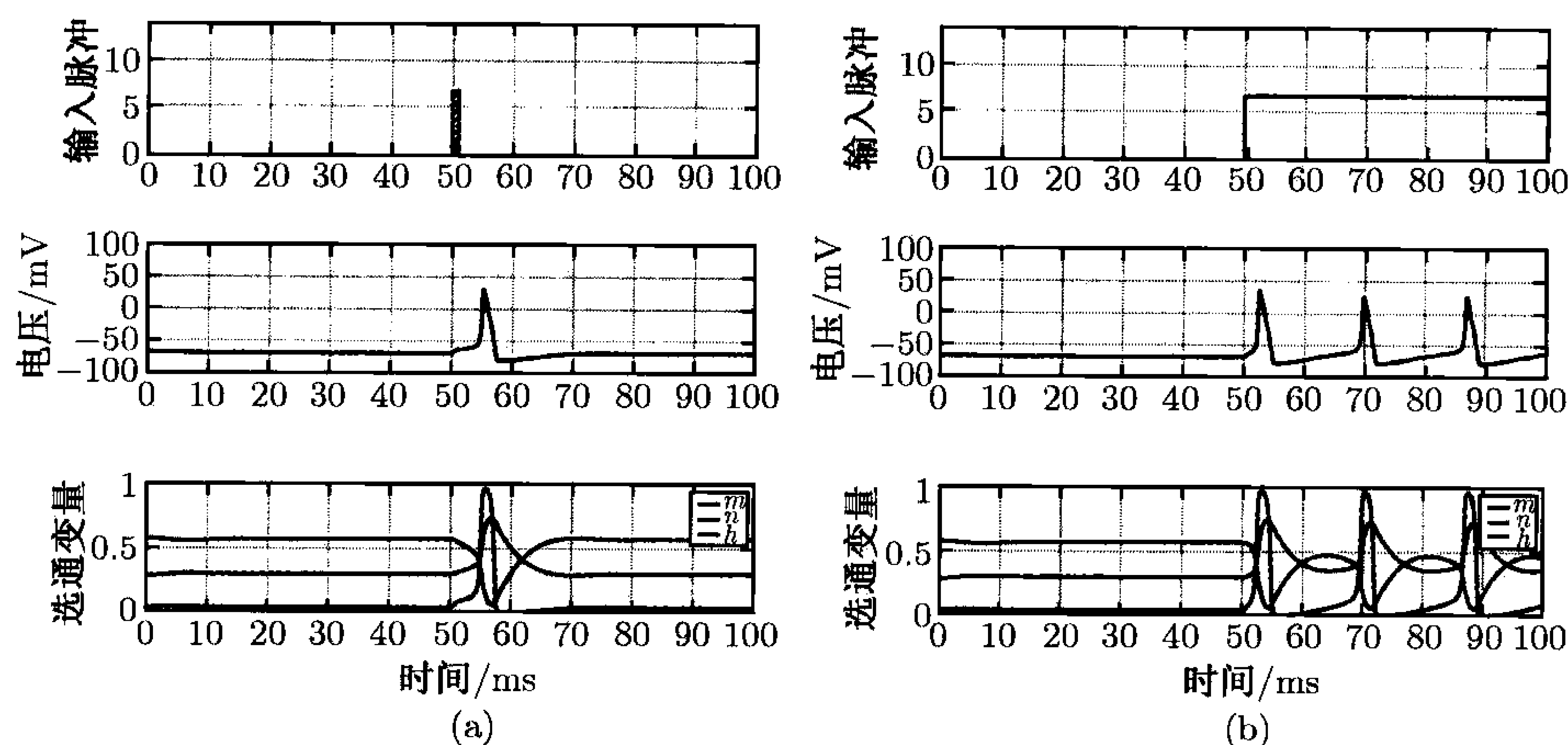


图 6-16　Hodgkin-Huxiely 程序的屏幕截图: (a) 在时间 50 微秒处、1 微秒期间的大小为 $I_{输入} = 7\mu A$ 的方波输入使模型神经元立即发射 (b) 以 $I_{输入} = 7\mu A$ 的持续平方波使得模型神经元周期性地发射

虽然 Lorenz 方程是确定的而且相当简单 (几乎线性), 但因为轨道十分复杂, 所以 Lorenz 方程是一个重要的例子. 对于复杂性的解释类似于双摆或三体问题: 即对初始条件的敏感依赖性. 计算机问题 8 和问题 9 研究了这种所谓混沌吸引子的敏感依赖性.

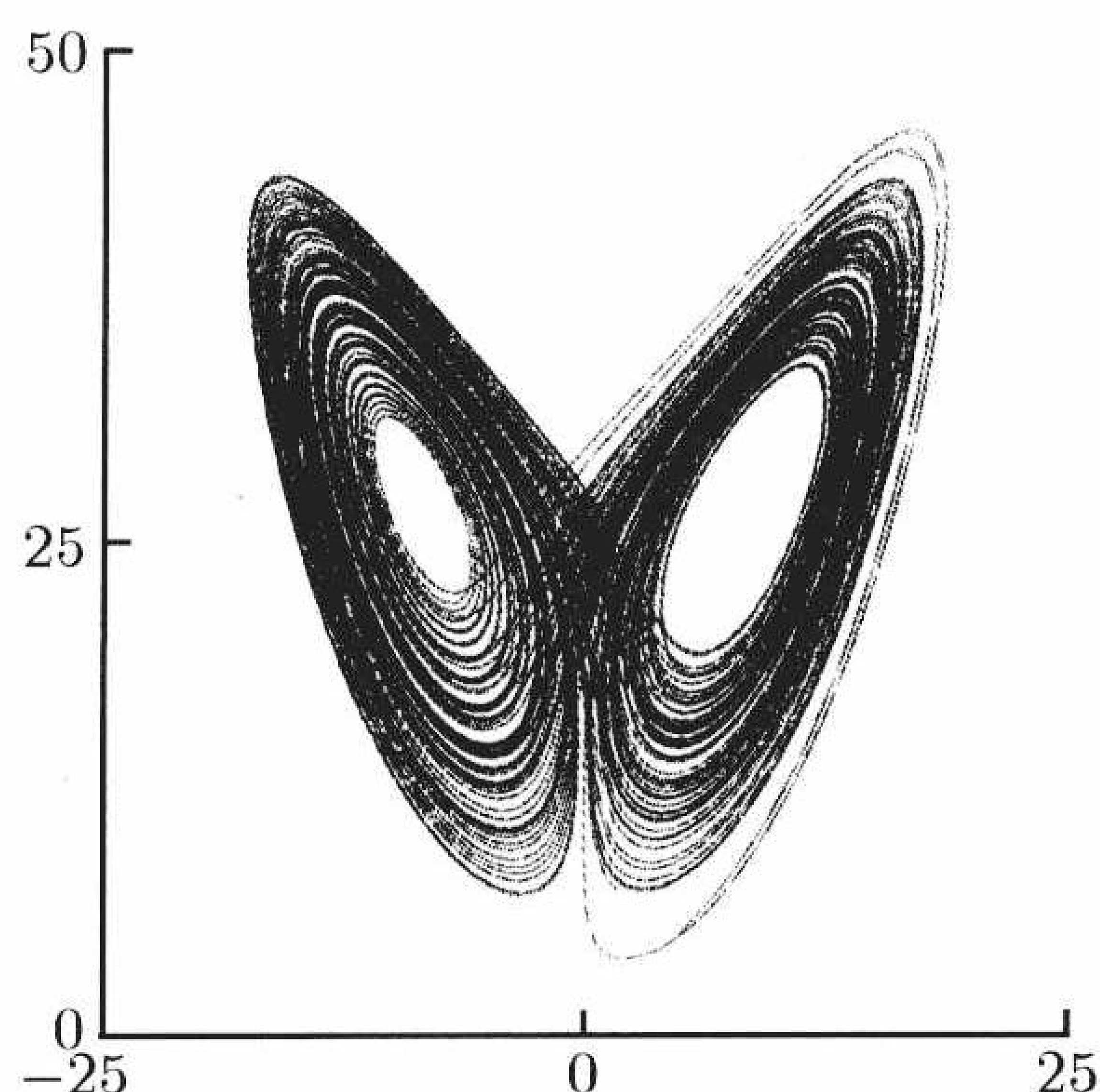


图 6-15 Lorenz 方程 (6.53) 的投影到 xz 平面上的一条轨道：取参数为 $s=10, r=28, b=\frac{8}{3}$

习题 6.4

1. 用中点方法求解以下初值问题：

(a) $y'=t$; **(b)** $y'=t^2y$; **(c)** $y'=2(t+1)y$;

(d) $y'=5t^4y$; **(e)** $y'=1/y^2$; **(f)** $y'=t^3/y^2$.

初始条件是 $y(0)=1$. 取步长 $h=\frac{1}{4}$, 在区间 $[0,1]$ 上用中点方法计算近似解. 与在习题 6.1.3 中求得的准确解进行比较, 并求 $t=1$ 处的整体截断误差.

2. 对以下初值问题执行习题 1 的步骤：

(a) $y'=t+y$; **(b)** $y'=t-y$; **(c)** $y'=4t-2y$.

初始条件是 $y(0)=0$. 准确解在习题 6.1.4 中求得.

3. 对习题 1 中的初值问题, 用 4 阶 Runge-Kutta 方法求解. 取步长 $h=\frac{1}{4}$, 计算区间 $[0,1]$ 上的近似解. 与习题 6.1.3 中求得的准确解进行比较, 并求 $t=1$ 处的整体截断误差.

4. 对习题 2 中的初值问题, 执行习题 3 的步骤.

5. 对任意 $\alpha\neq 0$, 证明方法 (6.49) 是二阶的.

6. 考虑初值问题 $y'=\lambda y$, 其解为 $y(t)=y_0e^{\lambda t}$. **(a)** 对这个微分方程用 RK4 通过 w_0 来计算 w_1. **(b)** 通过令 $w_0=y_0=1$ 并确定 y_1-w_1 来计算局部截断误差. 证明局部截断误差的大小是 $O(h^5)$, 就像 4 阶方法所预期的那样.

7. 假设右端 $f(t,y)=f(t)$ 不依赖于 y. 证明在 4 阶 Runge-Kutta 方法中 $s_2=s_3$, 并且 RK4 等价于积分 $\int_{t_i}^{t_i+h} f(s)\mathrm{d}s$ 的 Simpson 法则.

计算机问题 6.4

1. 在 $[0,1]$ 中步长 $h=0.1$ 的网格上, 对习题 1 中的初值问题用中点方法求解. 在每一步打印 t 值、近似解及整体截断误差的表格.

2. 在 $[0,1]$ 中步长 $h=0.1$ 的网格上, 对习题 1 中的初值问题用 4 阶 Runge-Kutta 方法求解, 在每一步打印 t 值、近似解及整体截断误差的表格.

3. 执行计算机问题 2 中的步骤, 但是对步长 $h=0.1, 0.05, 0.025$ 画出近似解和真解.

4. 对习题 2 中的方程执行计算机问题 2 中的步骤.
5. 对习题 1 中的初值问题, 像在图 6-4 中那样, 把 RK4 在 $t=1$ 处的整体误差画作 h 的函数.
6. 考虑带默认参数的 Hudgkin-Huxley 方程 (6.52). (a) 以微安为单位, 尽可能精确地求出形成 1 微秒脉冲尖峰的最小临界值. (b) 假如脉冲是 5 微秒长, 答案改变吗? (c) 用脉冲的形状进行试验. 一个所围面积相等的三角形脉冲与正方形脉冲造成相同的影响吗? (d) 对常数持续输入讨论临界值的存在性.
7. 修改 MATLAB程度`orbit.m` , 用 4 阶 Runge-Kutta 方法, 取步长 $h=0.001$, 求 Lorenz 方程的解. 画出满足初始条件 $(x_0,y_0,z_0)=(5,5,5)$ 的轨道.
8. 通过以下两个从很接近的初始条件得到的两条轨道来估计 Lorenz 方程的条件作用, 考虑初始条件 $(x,y,z)=(5,5,5)$ 及离第一个初始条件距离为 $\Delta=10^{-5}$ 的另一个初始条件. 取步长 $h=0.001$, 用 4 阶 Runge-Kutta 方法计算这两条轨道, 并且计算 $t=0$ 和 $t=20$ 时间单位处的误差放大因子.
9. 如同在计算机问题 8 中那样, 观察初始条件十分接近的 Lorenz 方程的两条轨道. 对每一条轨道构造一个由 0 和 1 组成的二进制符号序列: 在图 6-17 中, 如果轨道经过 "负 x" 圈, 就是 0; 如果它经过正圈, 就是 1. 这两条轨道的符号序列在多少时间内是一致的?

实例检验 6　塔科马海峡大桥

最近 McKenna and Tuama[18] 提出了企图寻找塔科马海峡大桥事故的数学模型. 目的是解释通过严格垂直的外力作用, 扭曲或扭转的振荡是如何被放大的.

考虑宽 $2l$ 的桥面吊在两根悬挂着的缆索之间. 如图 6-18a 所示. 在这个模型中我们将考虑桥的二维薄片, 忽略桥的长度这一维, 这是因为我们仅仅对这边到那边的运动有兴趣. 静止状态时, 由于重力, 桥面吊在某一平衡高度; 令 y 表示桥面中心在这个平衡位置下面的距离.

Hooke 定律假设线性力, 这意味着缆索所用的回复力将正比于位移, 设 θ 是桥面与水平方向所成之角. 有两根分别从平衡位置延伸 $y-l\sin\theta$ 和 $y+l\sin\theta$ 的支承缆索. 假定黏性阻尼项正比于速度. 利用 Newton 定律 $F=ma$ 并用 K 表示 Hooke 常数, 关于 y 和 θ 的运动方程如下:

$$\begin{aligned} y'' &= -dy' - \left[\frac{K}{m}(y-l\sin\theta)+\frac{K}{m}(y+l\sin\theta)\right], \\ \theta'' &= -d\theta' + \frac{3\cos\theta}{l}\left[\frac{K}{m}(y-l\sin\theta)-\frac{K}{m}(y+l\sin\theta)\right]. \end{aligned}$$

然而, Hooke 定律是为弹簧设计的, 不管压缩还是拉伸弹簧, 回复力大体是相等的. Mckenna 和 Tuama 假设在拉伸时拉回的力比在压缩时推回用力更多 (把弹簧想象成极端的例子). 他们用一种非线性 $f(y)=(k/a)(\mathrm{e}^{ay}-1)$ 代替 Hooke 定律中的回复力 $f(y)=ky$. 如图 6-18b 所示. 这两个函数在 $y=0$ 处都有相同的斜率 K; 但是对于非线性力, 正的 y(拉伸的缆索) 造成的回复力能比负的 y(松弛的缆索) 相应的回复力更强. 在前面的方程中进行这种代替便得到

$$\begin{aligned} y'' &= -dy' - \frac{K}{ma}\left[\mathrm{e}^{a(y-l\sin\theta)}-1+\mathrm{e}^{a(y+l\sin\theta)}-1\right], \\ \theta'' &= -d\theta' + \frac{3\cos\theta}{l}\frac{K}{ma}\left[\mathrm{e}^{a(y-l\sin\theta)}-\mathrm{e}^{a(y+l\sin\theta)}\right]. \end{aligned} \tag{6.54}$$

当方程成立时, 状态 $y = y' = \theta = \theta' = 0$ 就是一种平衡. 现在话题转向风. 把受力项 $0.2W\sin\omega t$ 加到 y 方程的右端, 这里 W 是风速 (千米/小时). 这就对桥加上了严格垂直的振荡.

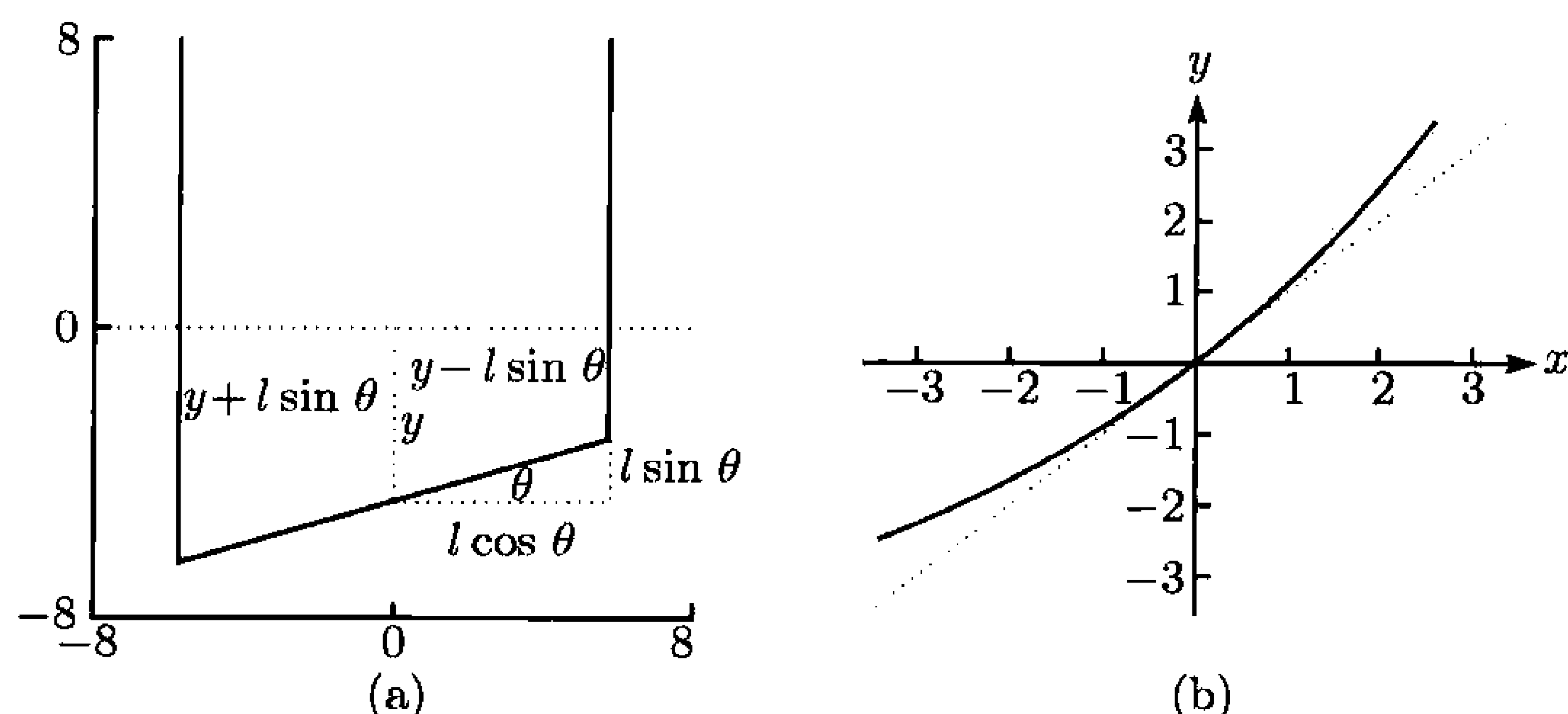


图 6-18 塔科马海峡大桥的 McKenna-Tuama 模型示意图: (a) y 表示桥面中心到它的平衡位置的距离, 桥面与水平的方向夹角为 θ; (b) 指数型 Hooke 定律曲线 $f(y) = (k/a)(e^{ay} - 1)$

对物理常数可以做出有用的估计. 一英尺长的桥面质量大约是 2 500 千克. 弹簧常数 K 已被估计为 1 000 牛顿, 桥面的宽大约 12 米. 对于这种模型, 阻尼系数设为 $d = 0.01$, Hooke 非线性系数 $a = 0.2$. 坍塌前不久, 观察者注意到在一分钟内桥垂直振荡 38 次: 取 $\omega = 2\pi(38/60)$. 这些系数仅仅是一种猜测, 但是它们足以显示与大桥最后振荡的事故照片相匹配的运动的范围. 运行模型 (6.54) 的 MATLAB 代码如下:

```
%Program 6.6 Animation program for bridge using IVP solver
%Inputs: int=[a b] time interval,
%ic=[y(1,1) y(1,2) y(1,3) y(1,4)],
%h=stepsize, p=steps per point plotted
%Calls a one-step method such as trapstep.m
%Example usage: tacoma([0 1000],[1 0 0.001 0],.04,3)
function tacoma(inter,ic,h,p)
clf                             % clear figure window
a=inter(1);b=inter(2);n=ceil((b-a)/(h*p)); % plot n points
y(1,:)=ic;                      % enter initial conds in y
t(1)=a;len=6;
set(gca,'XLim',[-8 8],'YLim',[-8 8], ...
   'XTick',[-8 0 8],'YTick',[-8 0 8], ...
   'Drawmode','fast','Visible','on','NextPlot','add');
cla;                            % clear screen
axis square                     % make aspect ratio 1-1
road=line('color','b','LineStyle','-','LineWidth',5,...
    'erase','xor','xdata',[],'ydata',[]);
lcable=line('color','r','LineStyle','-','LineWidth',1,...
    'erase','xor','xdata',[],'ydata',[]);
rcable=line('color','r','LineStyle','-','LineWidth',1,...
    'erase','xor','xdata',[],'ydata',[]);
for k=1:n
```

```
    for i=1:p
      t(i+1)=t(i)+h;
      y(i+1,:)=trapstep(t(i),y(i,:),h);
    end
    y(1,:)=y(p+1,:);t(1)=t(p+1);
    z1(k)=y(1,1);z3(k)=y(1,3);
    c=len*cos(y(1,3));s=len*sin(y(1,3));
    set(road,'xdata',[-c c],'ydata',[-s-y(1,1) s-y(1,1)])
    set(lcable,'xdata',[-c -c],'ydata',[-s-y(1,1) 8])
    set(rcable,'xdata',[c c],'ydata',[s-y(1,1) 8])
    drawnow; pause(h)
end

function y=trapstep(t,x,h)
%one step of the Trapezoid Method
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;

function ydot=ydot(t,y)
len=6;a=0.2; W=80; omega=2*pi*38/60;
a1=exp(a*(y(1)-len*sin(y(3))));
a2=exp(a*(y(1)+len*sin(y(3))));
ydot(1)=y(2);
ydot(2)=-0.01*y(2)-0.4*(a1+a2-2)/a+0.2*W*sin(omega*t);
ydot(3)=y(4);
ydot(4)=-0.01*y(4)+1.2*cos(y(3))*(a1-a2)/(len*a);
```

建议习题

1. 在风速 W=80 千米/小时及初始条件 $y = y' = \theta' = 0, \theta = 0.001$ 下运行`tacoma.m`. 假如 θ 的小干扰衰减, 那么桥在扭转方面是稳定的, 如果它们的增长远远超过原来的大小就不稳定. 对于 W 的这种值会出现哪一种情形?

2. 用 4 阶 Runge-Kutta 方法代替梯形方法来提高精度. 并加入新的图像视窗画出 $y(t)$ 和 $\theta(t)$.

3. 对 W=50 千米/小时, 方程组在扭转方面是稳定的. 对于小的初始角求放大因子, 即, 设 $\theta(0) = 10^{-3}$, 求最大角 $\theta(t)(0 \leqslant t < \infty)$ 对 $t(0)$ 的比. 对初始角 $\theta(0) = 10^{-4}, 10^{-5}, \cdots$ 放大因子是否近似地一致?

4. 求最小风速 W, 使得对于小干扰 $\theta(0) = 10^{-3}$ 有 100 或者更大的放大因子. 对这个 W 能定义一致的放大因子吗?

5. 设计并执行一种方法来计算第 4 步中的最小风速, 使其在 0.5×10^{-3} 千米/小时内. 可以用第 1 章中的一种方程解法.

6. 用几种较大的 W 值试一下. 所有非常小的初始角最终是否发展到灾难性的大小?

这个项目是实验数学的一个例子. 方程太难以致不能求得封闭形式的解, 甚至也难以证

明关于它的定性的结论. 有了可靠的常微分方程解法, 我们可以对各种参数设置产生数值轨道, 以说明适用于这个模型的现象的类别. 运用这种方式, 微分方程模型就可以预测科学和工程问题中的未来表现并弄清楚其中的机理.

6.5 变步长方法

直到现在, 在常微分方程解法的执行过程中, 步长 h 一直作为常数. 但是没有理由要求 h 在解的过程中不改变. 要改变步长的原因是为了求得运动在变化慢的时段和变化快的时段之间的解. 使得固定步长充分小以精确地追踪快速变化意味着其余的解是极慢地求得的.

本节讨论控制常微分方程解法的步长的策略. 最常见的逼近用两种不同阶的解法, 它们叫做嵌入对.

6.5.1 嵌入 Runge-Kutta 对

变步长方法的关键思想是控制由目前步长所产生的误差. 使用者设置一个适合目前步长的误差容限. 然后方法被设计为 (1) 假如超出误差容限, 则否决这一步并减小步长, 或者 (2) 假如符合误差容限, 则接受这一步然后选取适合下一步的步长 h. 关键的要求是用某种方法近似计算在每一步产生的误差. 首先假设我们已找到这样一种方法并解释如何改变步长.

改变步长的最简单方法是根据目前的误差, 对步长加倍或减半. 把误差估计 e_i 或相对误差估计 $\frac{e_i}{|w_i|}$ 与误差容限进行比较 (这里及本节其余部分, 我们将假定求解的常微分方程组包括一个方程, 把本节中的思想推广到高维是相当容易的). 假如误差容限不满足, 这一步就要用新步长等于 $\frac{h_i}{2}$ 来重复. 假如误差容限满足得很好, 譬如说误差小于误差容限的 $\frac{1}{10}$, 那么在接受这一步后, 下一步步长加倍.

用这种方法, 步长将被自动调节到这样一种大小：它保持 (相对) 局部截断误差接近使用者所要求的水平. 根据实际背景需要无论使用绝对误差或相对误差, 一种一般目的的好的技巧是用混合 $\frac{e_i}{\max(|w_i|,\theta)}$ 与误差容限进行比较, 这里常数 $\theta > 0$ 以免非常小的 w_i 值.

更复杂的方法是根据常微分方程解法的阶的知识选取适当的步长. 假定这种解法有 p 阶, 那么局部截断误差 $e_i = O(h^{p+1})$. 设 T 是使用者在每一步所允许的相对误差容限. 这就意味着目标是保证 $\frac{e_i}{|w_i|} < T$.

假定目标 $\frac{e_i}{|w_i|} < T$ 满足, 那么采用这一步, 并且需要下一步的新步长. 假设

$$e_i \approx ch_i^{p+1} \tag{6.55}$$

对某个常数 c 成立, 最佳符合误差容限的步长 h 满足

$$T|w_i| = ch^{p+1}. \tag{6.56}$$

从方程 (6.55) 和 (6.66) 解出 h 得到

$$h_* = 0.8 * \left(\frac{T|w_i|}{e_i}\right)^{\frac{1}{p+1}} h_i, \tag{6.57}$$

这里, 为了谨慎, 我们已加了一个安全因子 0.8. 于是下一个步长将取为 $h_{i+1} = h_*$.

另一方面, 假如相对误差没有满足目标 $\frac{e_i}{|w_i|} < T$, 那么作为第二次尝试, 取 h_i 为 h_*. 因为有安全因子, 这应该是足够的. 但是, 假如第二次尝试也没有满足这个目标, 那么步长就简单地减半. 继续这种方法有到目标达到为止. 正如为了更一般的目的所叙述的, 相对误差应该被 $\frac{e_i}{\max(|w_i|,\theta)}$ 所代替.

上述中, 无论是简单的方法还是复杂的方法, 都以某种方式异常地依赖目前步的常微分方程解的误差 $e_i = |w_{i+1} - y_{i+1}|$ 的估计. 一个重要的约束是得到这种估计不需要大量额外的计算.

得到这种误差估计的用得最广泛的方法是采用一种与感兴趣的常微分方程解法平行的高阶常微分方程解法. 这种高阶方法对 w_{i+1} (称它为 z_{i+1}) 的估计将比原来的 w_{i+1} 更为精确, 所以用差

$$e_i \approx |z_{i+1} - w_{i+1}| \tag{6.58}$$

作为从 t_i 到 t_{i+1} 的当前步的一种误差估计.

按照这种思想已经建立了 Runge-Kutta 方法的几种 "对": 一个 p 阶而另一个是 $p+1$ 阶的, 它们共享必需的计算 (没有大量额外的计算). 在这种方法中, 步长控制的额外代价保持很低. 这种 "对" 常常称为**嵌入 Runge-Kutta 对**.

例 6.19 RK2/3, 一个 Runge-Kutta 2 阶/3 阶嵌入对的例题.

显式梯形方法能与 3 阶 Runge-Kutta 方法配对, 得到适于步长控制的一种嵌入对. 令

$$\begin{aligned} w_{i+1} &= w_i + h\frac{s_1 + s_2}{2}, \\ z_{i+1} &= w_i + h\frac{s_1 + 4s_3 + s_2}{6}, \end{aligned}$$

其中

$$\begin{aligned} s_1 &= f(t_i, w_i), \\ s_2 &= f(t_i + h, w_i + hs_1), \\ s_3 &= f\left(t_i + \frac{1}{2}h, w_i + \frac{1}{2}h\frac{s_1 + s_2}{2}\right). \end{aligned}$$

在前面的方程中,w_{i+1} 是梯形步, 而 z_{i+1} 表示一种 3 阶方法, 它需要后面表出的 3 个 Runge-Kutta 级. 这种 3 阶方法恰巧把数值积分中的 Simpson 法则应用到了微分方程的内容中. 从这两个常微分方程解法中, 通过两个近似解相减就能够求得误差估计:

$$e_i \approx |w_{i+1} - z_{i+1}| = \left| h\frac{s_1 - 2s_3 + s_2}{3} \right|. \tag{6.59}$$

使用这种对局部截断误差的估计, 就可以执行前面描述的任一种步长控制草案.

尽管对于 w_{i+1} 来说步长草案已经确定, 但它对前进一步使用较高阶的近似 z_{i+1} 更有意义, 因为它是通用的. 这就叫做**局部外推**(local extrapolation).

例 6.20 Bogacki-Shampine 2 阶/3 阶嵌入对.

MATLAB在它的ode23命令中使用不同的嵌入对. 令

$$\begin{aligned} s_1 &= f(t_i, w_i), \\ s_2 &= f\left(t_i + \frac{1}{2}h, w_i + \frac{1}{2}hs_1\right), \\ s_3 &= f\left(t_i + \frac{3}{4}h, w_i + \frac{3}{4}hs_2\right), \\ z_{i+1} &= w_i + \frac{h}{9}(2s_1 + 3s_2 + 4s_3), \\ s_4 &= f(t + h, z_{i+1}), \\ w_{i+1} &= w_i + \frac{h}{24}(7s_1 + 6s_2 + 8s_3 + 3s_4). \end{aligned} \tag{6.60}$$

可以验证 z_{i+1} 是 3 阶近似, 而 w_{i+1} 尽管有 4 级但只是二阶的. 步长控制需要的误差估计是

$$e_i = |z_{i+1} - w_{i+1}| = \frac{h}{72}|-5s_1 + 6s_2 + 8s_3 - 9s_4|. \tag{6.61}$$

注意, 只要 s_4 被接受, 在下一步 s_4 变成 s_1, 所以没有浪费的步骤: 3 阶 Runge-Kutta 方法无论如何至少需要 3 级. 这种二阶方法的设计叫 FSAL(First Same As Last). ◀

6.5.2 4/5 阶方法

例 6.21 Runge-Kutta-Fehlberg 4 阶/5 阶嵌入对.

$$\begin{aligned} s_1 &= f(t_i, w_i), \\ s_2 &= f\left(t_i + \frac{1}{4}h, w_i + \frac{1}{4}hs_1\right), \\ s_3 &= f\left(t_i + \frac{3}{8}h, w_i + \frac{3}{32}hs_1 + \frac{9}{32}hs_2\right), \\ s_4 &= f\left(t_i + \frac{12}{13}h, w_i + \frac{1\,932}{2\,197}hs_1 - \frac{7\,200}{2\,197}hs_2 + \frac{7\,296}{2\,197}hs_3\right), \end{aligned}$$

$$
\begin{aligned}
s_5 &= f\left(t_i + h, w_i + \frac{439}{216}hs_1 - 8hs_2 + \frac{3\,680}{513}hs_3 - \frac{845}{4\,104}hs_4\right), \\
s_6 &= f\left(t_i + \frac{1}{2}h, w_i - \frac{8}{27}hs_1 + 2hs_2 - \frac{3\,544}{2\,565}hs_3 + \frac{1\,859}{4\,104}hs_4 - \frac{11}{40}hs_5\right), \\
w_{i+1} &= w_i + h\left(\frac{25}{216}s_1 + \frac{1\,408}{2\,565}s_3 + \frac{2\,197}{4\,104}s_4 - \frac{1}{5}s_5\right), \\
z_{i+1} &= w_i + h\left(\frac{16}{135}s_1 + \frac{6\,656}{12\,825}s_3 + \frac{28\,561}{56\,430}s_4 - \frac{9}{50}s_5 + \frac{2}{55}s_6\right).
\end{aligned} \tag{6.62}
$$

可以验证 z_{i+1} 是 5 阶近似, 而 w_{i+1} 是 4 阶. 步长控制所需要的误差估计是

$$
e_i = |z_{i+1} - w_{i+1}| = h\left|\frac{1}{360}s_1 - \frac{128}{4\,275}s_3 - \frac{2\,197}{75\,240}s_4 + \frac{1}{50}s_5 + \frac{2}{55}s_6\right|. \tag{6.63}
$$

◀

Runge-Kutta-Fehlberg 方法 (RKF45) 是目前最著名的变步长单步方法. 其实现起来很简单, 由前面的公式给出. 使用者必须设置一个相对误差容限 T 和初始步长 h. 计算 w_1, z_1 和 e_0 之后, 对 $i=0$ 检查相对误差测试

$$
\frac{e_i}{|w_i|} < T. \tag{6.64}
$$

假如成功, 就把新的 w_1 代替为局部外推式 z_1, 并且程序进入下一步. 另一方面, 假如相对误差测试 (6.64) 失败, 那这一步再尝试一遍, 这时的步长 h 由 (6.57) 在 $p=4$(即产生 w_i 的方法的阶) 给定.[如果重复失败 (这是不大可能的), 就把步长减半直到成功.] 不管何种情形, 下一步的步长 h_1 应该从 (6.57) 中计算得到.

例 6.22 Dormand-Prince 4 阶/5 阶嵌入对.

$$
\begin{aligned}
s_1 &= f(t_i, w_i), \\
s_2 &= f\left(t_i + \frac{1}{5}h, w_i + \frac{1}{5}hs_1\right), \\
s_3 &= f\left(t_i + \frac{3}{10}h, w_i + \frac{3}{40}hs_1 + \frac{9}{40}hs_2\right), \\
s_4 &= f\left(t_i + \frac{4}{5}h, w_i + \frac{44}{45}hs_1 - \frac{56}{15}hs_2 + \frac{32}{9}hs_3\right), \\
s_5 &= f\left(t_i + \frac{8}{9}h, w_i + h\left(\frac{19\,372}{6\,561}s_1 - \frac{25\,360}{2\,187}s_2 + \frac{64\,448}{6\,561}s_3 - \frac{212}{729}s_4\right)\right), \\
s_6 &= f\left(t_i + h, w_i + h\left(\frac{9\,017}{3\,168}s_1 - \frac{355}{33}s_2 + \frac{46\,732}{5\,247}s_3 + \frac{49}{176}s_4 - \frac{5\,103}{18\,656}s_5\right)\right), \\
z_{i+1} &= w_i + h\left(\frac{35}{384}s_1 + \frac{500}{1\,113}s_3 + \frac{125}{192}s_4 - \frac{2\,187}{6\,784}s_5 + \frac{11}{84}s_6\right), \\
s_7 &= f(t_i + h, z_{i+1}),
\end{aligned}
$$

$$w_{i+1} = w_i + h\left(\frac{5\ 179}{57\ 600}s_1 + \frac{7\ 571}{16\ 695}s_3 + \frac{393}{640}s_4 - \frac{92\ 097}{339\ 200}s_5 + \frac{187}{2\ 100}s_6 + \frac{1}{40}s_7\right). \tag{6.65}$$

可以验证 z_{i+1} 是 5 阶近似, 而 w_{i+1} 是 4 阶. 步长控制所需要的误差估计是

$$e_i = |z_{i+1} - w_{i+1}| = h\left|\frac{71}{57\ 600}s_1 - \frac{71}{16\ 695}s_3 + \frac{71}{1\ 920}s_4 - \frac{17\ 253}{339\ 200}s_5 + \frac{22}{525}s_6 - \frac{1}{40}s_7\right|. \tag{6.66}$$

再次用了局部外推, 意味着这一步由 z_{n+1} 向前推进而不是 w_{i+1}. 注意, 事实上不需要计算 w_{i+1}: 对于误差的控制仅需要 e_i. 像 Bogacki-Shampine 方法一样, 这是一种 FSAL 方法, 因为如果 s_7 能被接受, 在下步中 s_7 变成 s_1. 这里没有多余的步骤, 可以证明 5 阶 Runge-Kutta 方法至少需要 6 级. ◀

MATLAB命令ode45用 Dormand-Prince 嵌入对以及步长控制 (大致如前所述). 使用者可以把相对误差容限 T 设置成所需要的. 可以把微分方程的右端规定在函数文件中. 例如定义包括二行的文件f.m

```
function y=f(t,y)
y=t*y+t^3;
```

命令

```
>> opts=odeset('RelTol',1e-4,'Refine',1,'MaxStep',1);
>> [t,y]=ode45('f',[0 1],1,opts);
```

将解出例 6.1 中带有初始条件 $y_0 = 1$ 的初值问题, 而且相对误差容限 $T = 0.000\ 1$. 假如没有设置参数RelTol, 就使用默认的 0.001. 注意输入到ode45的函数 f 必须是两个变量的函数, 这里是 t 和 y, 即使其中一个没有出现在函数的定义中. 可以在不出现函数文件的情况下, 通过定义函数 f“内联” 运行这个命令.

```
>> [t,y]=ode45(inline('t*y+t^3','t','y'),[0 1],1,opts);
```

使用前面对这个问题设置的参数, ode45的输出结果见表 6-6.

表 6-6

步长	t_i	w_i	y_i	e_i
0	0.000 000 00	1.000 000 00	1.000 000 00	0.000 000 00
1	0.540 212 87	1.179 468 18	1.179 463 45	0.000 004 73
2	1.000 000 00	1.946 178 12	1.946 163 81	0.000 014 31

假如使用 10^{-6} 作为相对误差容限, 输出结果见表 6-7.

尽管步长是为 w_{i+1} 充分设计的, 但使用的是 z_{i+1} 而不是 w_{i+1}, 因为局部外推, 所以近似解更能满足相对误差容限. 这是我们能做得最好的. 假如有对 z_{i+1} 的误差估计, 我们就能使用它把步长调节得更好, 但是并没有这种估计. 还注意解精确地停在区间 $[0,1]$ 的末端, 这是因为 ode45 检测区间的末端并把步骤截短到必要

的程度.

表　6-7

步长	t_i	w_i	y_i	e_i
0	0.000 000 00	1.000 000 00	1.000 000 00	0.000 000 00
1	0.215 062 62	1.023 934 40	1.023 934 40	0.000 000 00
2	0.430 125 24	1.105 744 41	1.105 744 40	0.000 000 01
3	0.686 077 29	1.325 356 58	1.325 356 53	0.000 000 05
4	0.911 922 46	1.715 151 56	1.715 151 44	0.000 000 12
5	1.000 000 00	1.946 163 94	1.946 163 81	0.000 000 13

为了看清`ode45`如何进行它的步长选择, 我们必须使用`odeset`命令撤开几个基本的默认设置. 参数`Refine`一般会增加记录的解值的数量, 它比用方法计算得到的要多以作出更漂亮的图, 只要输出用于这个目的. 默认值是 4, 这造成必要的点数的 4 倍被输出. 参数`MaxStep`赋于步长 h 的上限, 为区间长度的 $\frac{1}{10}$. 对这两种参数用默认值意味着要用步长 $h = 0.1$, 在用因子 4 改进后, 解可能用 0.025 步长表出. 事实上, 运行没有指定输出变量的命令, 例如

```
>> opts=odeset('RelTol',1e-6);
>> ode45(inline('t*y+t^3','t','y'),[0 1],1,opts);
```

将使 MATLAB自动画出在常数步长 0.025 的网格上的解. 如图 6-19 所示.

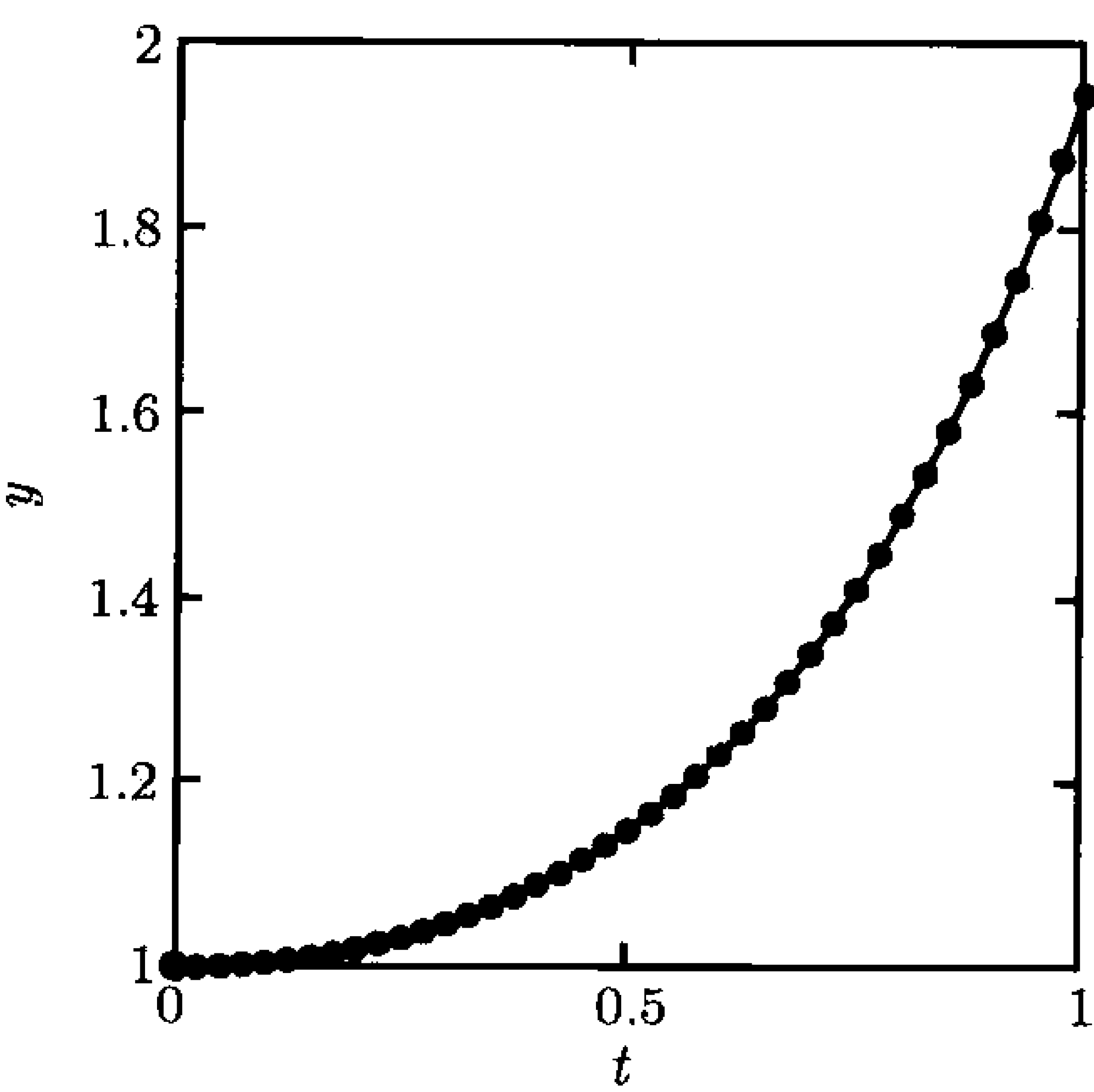


图 6-19　MATLAB的`ode45`命令: 计算出例 6.1 中的初值问题的解, 准确到 10^{-6}

虽然我们有意把变步长 Runge-Kutta 方法当作最好的常微分方程解法, 但也有一些方程类型它处理得不好. 下面是一个特别简单的例子:

例 6.23　用`ode45`求解初值问题

$$\begin{cases} y' = 10(1-y), \\ y(0) = \dfrac{1}{2}, \\ t \in [0, 100], \end{cases} \tag{6.67}$$

要求相对误差容限为 10^{-4}.

这能用下面 3 行 MATLAB代码完成.

```
>> opts=odeset('RelTol',1e-4);
>> [t,y]=ode45(inline('10*(1-y)','t','y'),[0 100],.5,opts);
>> length(t)

ans=        1241
>>
```

我们已经打印出步数, 因为它看似过大. 这个初值问题的解容易确定: $y(t) = 1 - e^{-10t}/2$. 对 $t > 1$, 这个解已经达到它的平衡位置 1, 精确到 4 位小数, 而且它决不会离开 1 更远. 应用小于 0.1 的平均步长,ode45依然缓慢地运动. 为什么如此保守的步长选择却得到一个理想的解?

通过观察图 6-20 中的ode45的输出, 部分答案就清楚了. 尽管这个解非常接近 1, 但算法连续地过调以企图近似得更好. 微分方程是 "刚性的", 这个术语将在 6.6 节中正式定义. 对于刚性方程, 数值解中的不同策略大大地增加了解的效率. 例如, 当使用一个 MATLAB的刚性解法时, 注意所需要的步骤的差别.

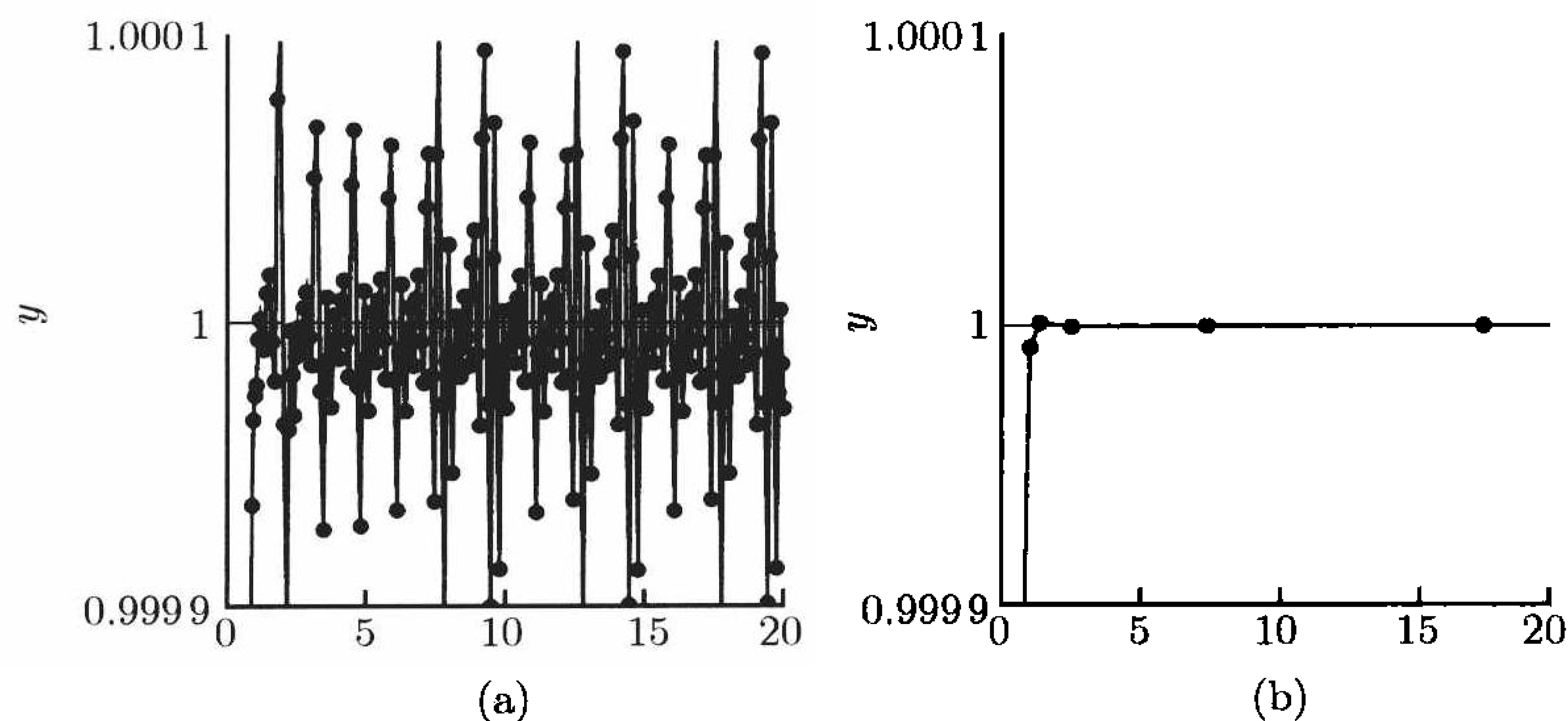


图 6-20 例 6.23 中的初值问题的数值解: (a) 使用ode45需要每单位时间超过 10 步使相对误差容限在 10^{-4} 之内; (b) 使用ode23, 需要的步数要少得多

```
>> opts=odeset('RelTol',1e-4);
>> [t,y]=ode23s(inline('10*(1-y)','t','y'),[0 100],.5,opts);
>> length(t)
```

```
ans=
        39
>>
```

图 6-20b 画出了解法ode23s得到的解点. 为了保证数值解在误差容限之内, 只需要相对少的点. 6.6 节将研究如何建立处理这种困难类型的方法. ◀

计算机问题 6.5

1. 写出 RK23(例 6.19) 的 MATLAB程序, 并用来求在习题 6.1.3 中的初值问题的近似解, 要求在 [0,1] 的相对误差容限为 10^{-8}. 要求程序恰好在终点 $t=1$ 停止. 记录所用的最大步长和步数.
2. 把计算机问题 1 中的结果与 MATLAB的ode23用于同一问题的结果进行比较.
3. 对于 Runge-Kutta-Fehlberg 方法 RKF45, 执行计算机问题 1 中的步骤.
4. 把计算机问题 3 的结果与 MATLAB的ode45用于同一问题的结果进行比较.
5. 用 RKF45 的 MATLAB程序, 求习题 6.3.1 中的方程组的近似解, 要求在 [0,1] 中的相对误差容限是 10^{-6}. 记录所用的最大步长和步数.

6.6 隐式方法和刚性方程

到目前为止, 我们已经提出的微分方程的解法都是**显式的**(explicit), 即新的近似解 w_{i+1} 可以通过诸如 h, t_i 及 w_i 这些已知数据来显式地表示. 然而某些微分方程不适宜用显式方法, 我们首先来解释原因. 在例 6.23 中, 复杂的变步长解法似乎花费大部分精力用于过调在一个方向或另一个方向的准确解.

在比较简单的实际背景中, 能够较为容易地了解刚性现象. 因此, 我们从 Euler 方法开始.

例 6.24 对于例 6.23 使用 Euler 方法.

取步长 h, Euler 方法对于右端 $f(t,y)=10(1-y)$ 是

$$w_{i+1}=w_i+hf(t_i,w_i)=w_i+h(10)(1-w_i)=w_i(1-10h)+10h. \tag{6.68}$$

由于解是 $y(t)=1-\mathrm{e}^{-10t}/2$, 所以近似解最终必须趋于 1. 这里, 我们从第 1 章得到一些帮助. 注意 (6.68) 可以看成 $g(x)=x(1-10h)+10h$ 的不动点迭代. 这个迭代有不动点 $x=1$, 而且只要 $|g'(1)|=|1-10h|<1$, 它将是收敛的. 解这个不等式得到 $0<h<0.2$. 对任何较大的 h, 不动点将排斥附近的推测, 因此解将不可能是精确的. ◀

图 6-21 表明对例 6.24 的这种影响. 解非常理想: 一个吸引人的在 $y=1$ 处的平衡. 步长 $h=0.3$ 的一个 Euler 步难以求得平衡位置, 这是因为在 h 区间的起点和终点之间, 解附近的斜率变化很大. 这就造成数值解的溢出.

具有这种性质 (吸引解被附近快速改变的解所包围) 的微分方程称为**刚性的** (stiff). 这经常是方程组中多倍尺度的信号. 就数量而言, 它相应于微分方程的右端 f 关于变量 y 的线性部分是大的而且是负的 (对方程组而言, 这相应于线性部分的特征值是大的而且是负的). 这个定义有一点相对, 但这是刚性的本质: 越大而且越负, 为了避免过调那么步长就要越小. 对于例 6.24, 刚性是通过计算在平衡解 $y=1$ 处的 $\frac{\partial f}{\partial y}=-10$ 而度量的.

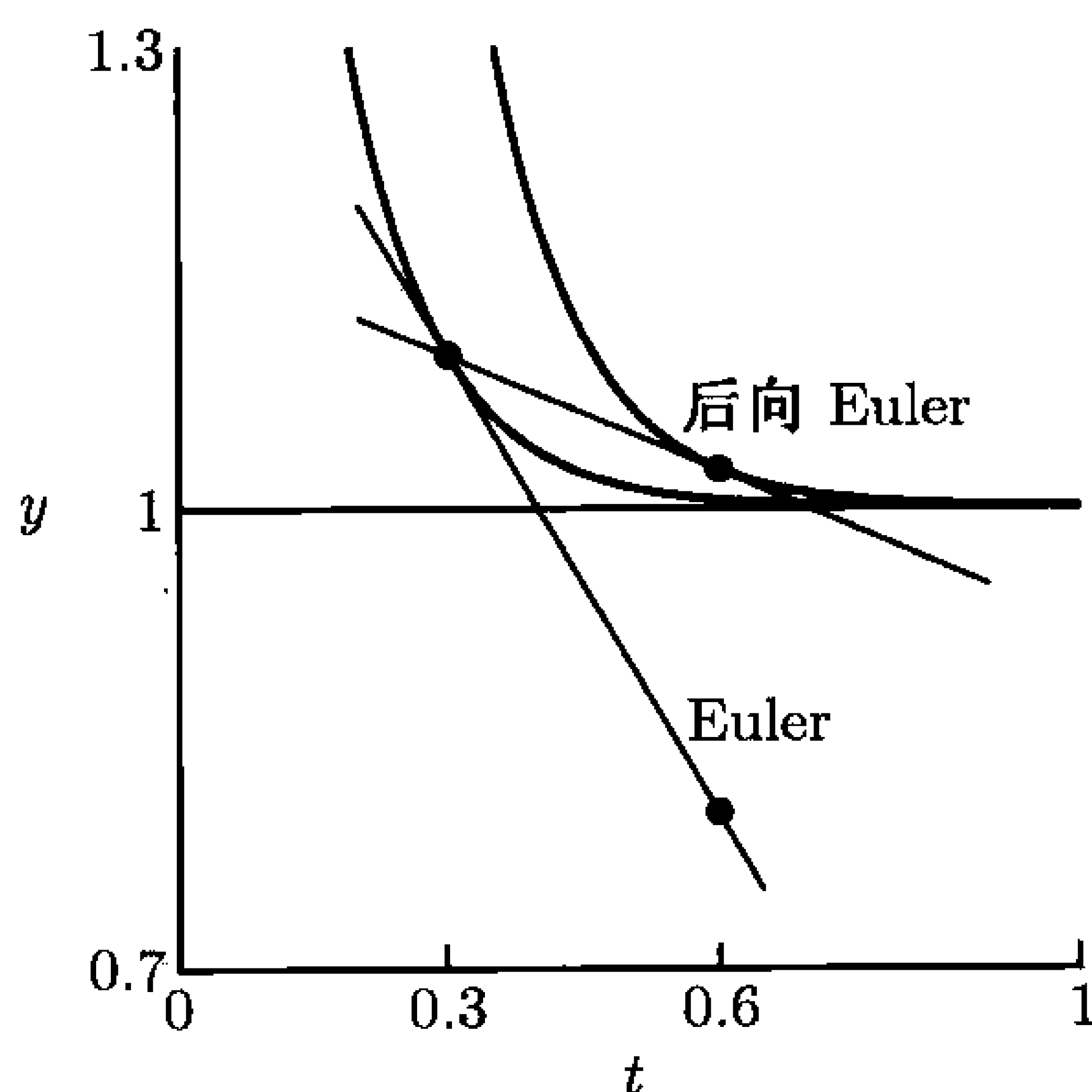


图 6-21 Euler 步和后向 Euler 步的比较. 例 6.23 中的微分方程是刚性的. 平衡解 $y=1$ 被其他大曲率 (斜率变化很快) 的解所包围. Euler 步过调, 而后向 Euler 步与系统的动态更一致

解图 6-21 所描绘的问题的一种方法是, 多多少少从区间 $[t_i, t_i+h]$ 的右边引入信息, 而不是仅仅依靠左边的信息. 这就促使我们得到以下 Euler 方法的变形.

后向 Euler 方法

$$\begin{aligned} w_0 &= y_0, \\ w_{i+1} &= w_i + hf(t_{i+1}, w_{i+1}). \end{aligned} \tag{6.69}$$

注意其中的区别: Euler 方法用左边点的斜率来通过区间, 后向 Euler 方法则用某种方法通过区间使得斜率在右边点上是正确的.

对这种改进必须付出代价. 后向 Euler 方法是我们的**隐式**(implicit) 方法的第一个例子, 即这种方法关于新的近似解 w_{i+1} 并不直接给出一个公式. 相反, 我们必须做一点工作来得到它. 对于例子 $y'=10(1-y)$, 后向 Euler 方法给出

$$w_{i+1} = w_i + 10h(1-w_{i+1}),$$

经过一些代数运算之后, 就能表示为

$$w_{i+1} = \frac{w_i + 10h}{1+10h}.$$

例如, 取 $h=0.3$, 后向 Euler 方法给出 $w_{i+1}=(w_i+3)/4$. 当作为不动点迭代 $w\to g(w)=\frac{w+3}{4}$ 时, 我们又能估计它的性质. 在 1 处有不动点, 并且 $g'(1)=\frac{1}{4}<1$, 这就证明了对平衡真解 $y=1$ 的收敛性. 与 $h=0.3$ 时的 Euler 方法不同, 数值解至少遵守正确的定性性质. 事实上, 注意不管步长 h 多么大, 后向 Euler 方法的解收敛于 $y=1$(习题 3).

在有刚性方程的情况下, 因为有像后向 Euler 方法这样的隐式方法的较好性质, 计算下一步值得多做一些额外的工作, 即使它不是明显可得到的. 对于求解 w_{i+1}, 例 6.24 并不具有挑战性, 这是由于微分方程是线性的, 因此, 为了求值有可能把原来的隐式公式改变成显式. 然而, 一般情形下这是不可能的, 因此我们需要用更间接的方法.

把隐式方法用于求解非线性方程, 我们必参阅第 1 章. 不动点迭代和 Newton 方法都经常用来求 w_{i+1}. 这意味着在循环推进微分方程内存在方程求解的循环. 下面的例子说明了如何做到这一点.

例 6.25　对初值问题

$$\begin{cases} y'=y+8y^2-9y^3, \\ y(0)=\dfrac{1}{2}, \\ t\in[0,3], \end{cases}$$

用后向 Euler 方法求解.

像前面的方程一样, 这个方程有一个平衡解 $y=1$. 偏导数 $\frac{\partial f}{\partial y}=1+16y-27y^2$ 在 $y=1$ 处等于 -10, 表明这个方程有一定程度的刚性. 类似于前面的例题, 对 h 将有一个上界, 这样 Euler 方法是成功的. 这就促使我们尝试后向 Euler 方法:

$$w_{i+1}=w_i+hf(t_{i+1},w_{i+1})=w_i+h(w_{i+1}+8w_{i+1}^2-9w_{i+1}^3).$$

这是关于 w_{i+1} 的非线性方程, 这是我们为了提出数值解而需要求解的. 再记 $z=w_{i+1}$, 我们必须解方程 $z=w_i+h(z+8z^2-9z^3)$, 或者关于未知量 z 求解

$$9hz^3-8hz^2+(1-h)z-w_i=0 \tag{6.70}$$

我们将用 Newton 方法来说明.

为了开始 Newton 方法, 需要一个初始估计. 能想到的两种选择是: 前面的近似 w_i 和 Euler 方法对于 w_{i+1} 的近似. 尽管由于 Euler 方法是显式的而使得后者容易达到, 但是, 如图 6-21 所示, 对于刚性问题它可能并不是最佳选择. 此时, 我们将用 w_i 作为初始估计.

对 (6.70) 配置 Newton 方法得到

$$z_{新}=z-\frac{9hz^3-8hz^2+(1-h)z-w_i}{27hz^2-16hz+1-h}. \tag{6.71}$$

求出 (6.71) 之后, 用 $z_{新}$ 代替 z, 并重复进行. 对于每一个后向 Euler 步, 执行 Newton 方法, 直到 $z_{新} - z$ 小于一个预设的容限 (比在近似微分方程解中产生的误差小).

图 6-22 显示对于两个不同步长的结果. 另外, 还显示了 Euler 方法的数值解. 很清楚, Euler 方法对于这种刚性问题而言 $h = 0.3$ 是太大了. 另一方面, 当 h 减到 0.15 时, 两种方法执行的水平大概相同.

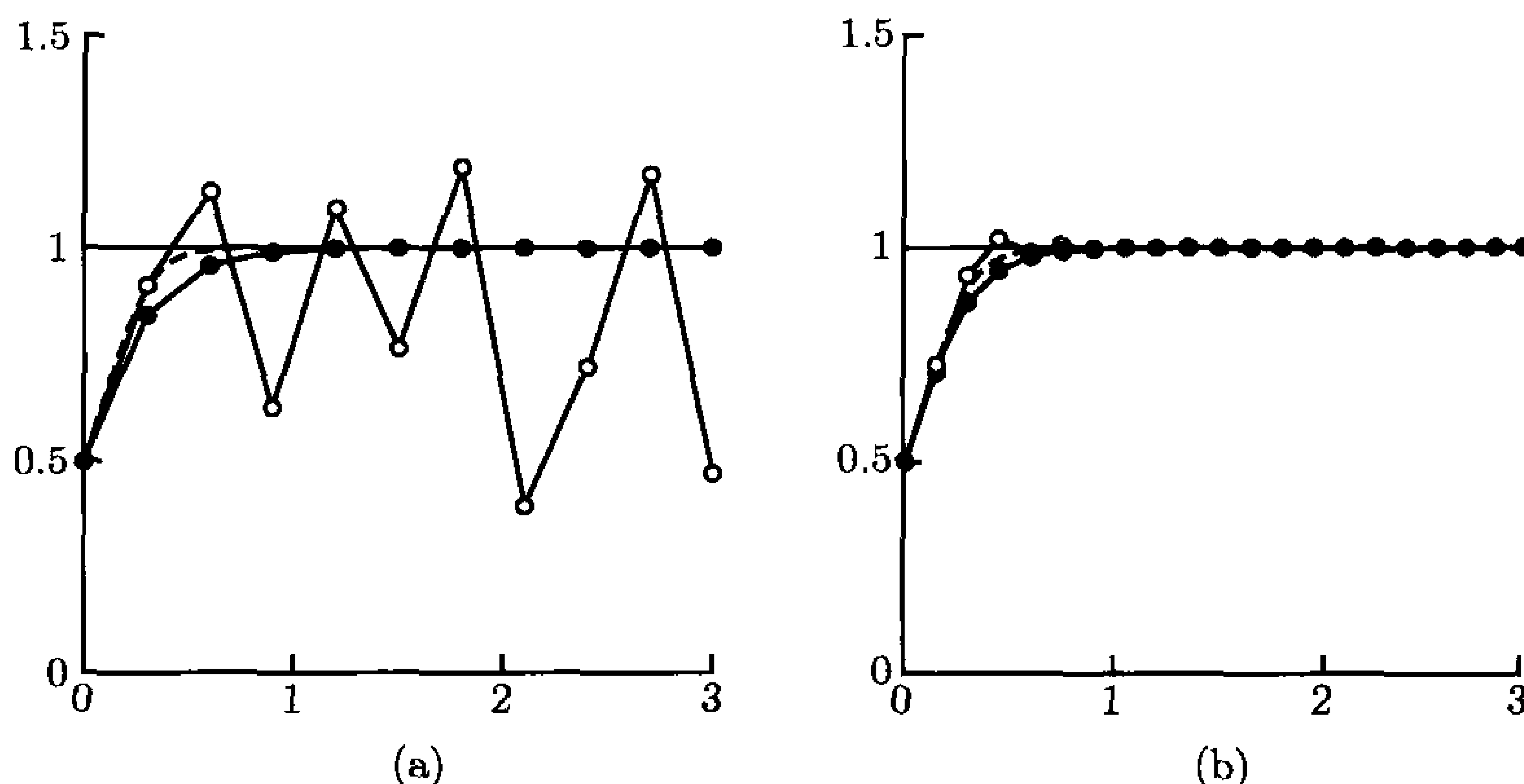


图 6-22 例 6.25 中初值问题的数值解. 真解是虚曲线, 空圆圈表示 Euler 方法近似解, 实圆圈表示后向 Euler 方法: (a) $h = 0.3$, (b)=0.15

像后向 Euler 方法这样的所谓刚性解法允许有足够的误差控制, 同时采用较大的步长, 提高了效率. MATLAB ode23s是一个有内置变步长策略的较高阶形式. ◀

习题 6.6

1. 用初始条件 $y(0) = 0$ 和步长 $h = \frac{1}{4}$, 在 [0,1] 区间计算后向 Euler 近似解. 通过与在习题 6.1.4 中求得的准确解比较, 求在 $t = 1$ 处的误差.
   (a) $y' = t + y$; (b) $y' = t - y$; (c) $y' = 4t - 2y$.
2. 求所有的平衡解及在平衡解处的 Jacobi 行列式的值. 方程是刚性的吗?
   (a) $y' = y - y^2$; (b) $y' = 10y - 10y^2$; (c) $y' = -10\sin y$.
3. 对习题 6.24 证明, 对每一种步长 h, 当 $t_i \to \infty$ 时后向 Euler 近似解收敛于平衡解 $y = 1$.
4. 考虑线性微分方程 $y' = ay + b(a < 0)$. (a) 求平衡解. (b) 对方程写出后向 Euler 方法. (c) 把后向 Euler 方法看做不动点迭代, 证明这种方法的近似解当 $t \to \infty$ 时将收敛于平衡解.

计算机问题 6.6

1. 把 Newton 方法作为一种解法, 对初值问题应用后向 Euler 方法. 近似解逼近哪一个平衡解? 应用 Euler 方法. 对 h 的什么近似范围, 可以用 Euler 方法成功地收敛于平衡解?

画出由后向 Euler 方法给出的近似解和由 Euler 方法用过分大的步长所给出的近似解.

(a) $\begin{cases} y' = y^2 - y^3, \\ y(0) = \frac{1}{2}, \\ t \in [0, 20]; \end{cases}$　(b) $\begin{cases} y' = 6y - 6y^2, \\ y(0) = \dfrac{1}{2}, \\ t \in [0, 20]. \end{cases}$

2. 对以下初值问题执行计算机问题 1 中的步骤:

(a) $\begin{cases} y' = 6y - 3y^2, \\ y(0) = \frac{1}{2}, \\ t \in [0, 20]; \end{cases}$　(b) $\begin{cases} y' = 10y^3 - 10y^4, \\ y(0) = \frac{1}{2}, \\ t \in [0, 20]. \end{cases}$

6.7　多 步 方 法

我们已研究过的 Runge-Kutta 族由单步方法组成, 意即最新步 w_{i+1} 是以微分方程以及前一步的值 w_i 为基础生成的. 这是初值问题本质的表现. 对此, 定理 6.2 保证从任意的 w_0 出发有唯一解.

多步方法提出一种不同的逼近: 用比前面一个 w_i 更多的知识去帮助产生下一步. 这将导出与单步方法具有同样阶的常微分方程解法, 但是许多必需的计算将被在求解过程中已经算得的值的插值所代替.

6.7.1　生成多步方法

作为第一个例子, 考虑以下两步方法:

Adams-Bashforth 两步方法

$$w_{i+1} = w_i + h\left[\frac{3}{2}f(t_i, w_i) - \frac{1}{2}f(t_{i-1}, w_{i-1})\right]. \tag{6.72}$$

而二阶中点方法

$$w_{i+1} = w_i + hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)\right)$$

每一步需要对常微分方程右端 f 的函数进行两次求值, 这种 Adams-Bashforth 两步方法每步仅需一次新的求值 (一次储存在前一步). 随后将看到, (6.72) 也是一个二阶方法. 因此, 多步方法能用较少的计算而达到相同的阶数: 一般每步仅计算一个函数值.

因为多步方法用到前面的多个 w 值, 它们需要帮助开始启动. s 步方法的起动阶段一般包括在能用多步方法之前, 用 w_0 产生 $s-1$ 个值 $w_1, w_2, \cdots, w_{s-1}$ 的一步方法. 为了开始, Adams-Bashforth 两步方法 (6.72) 需要 w_1 以及给定的初始条件 w_0. 以下 MATLAB 代码用梯形方法产生起动值 w_1. 命令`plot(t,y)`用来画出输出结果.

```
% Program 6.7 Multistep method
% Inputs: [inter(1),inter(2)] time interval,
%  ic=[y0]  initial condition,
%  h=stepsize, s=number of (multi)steps, e.g. 2 for 2-step method
% Output:time steps t, solution y
% Calls a multistep method such as ab2step.m
% Example usage: exmultistep ([0,1],1,0.05,2)
function [t,y]=exmultistep(inter,ic,h,s)
n=round((inter(2)-inter(1))/h);
% Start-up phase
y(1,:)=ic;t(1)=int(1);
for i=1:s-1                % start-up phase, using one-step method
  t(i+1)=t(i)+h;
  y(i+1,:)=trapstep(t(i),y(i,:),h);
  f(i,:)=ydot(t(i),y(i,:));
end
for i=s:n                  % multistep method loop
  t(i+1)=t(i)+h;
  f(i,:)=ydot(t(i),y(i,:));
  y(i+1,:)=ab2step(t(i),i,y,f,h);
end
function y=trapstep(t,x,h)
%one step of the Trapezoid Method from section 6.2
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;

function z=ab2step(t,i,y,f,h)
%one step of the Adams-Bashforth 2-step method
z=y(i,:)+h*(3*f(i,:)/2-f(i-1,:)/2);

function z=unstable2step(t,i,y,f,h)
%one step of an unstable 2-step method
z=-y(i,:)+2*y(i-1,:)+h*(5*f(i,:)/2+f(i-1,:)/2);

function z=weaklystable2step(t,i,y,f,h)
%one step of a  weakly-stable 2-step method
z=y(i-1,:)+h*2*f(i,:);

function z=ydot(t,y)  % IVP from section 6.1
z=t*y+t^3;
```

图 6-23a 表示在本章的初值问题 (6.5) 中用 Adams-Bashforth 方法的结果, 取步长 $h = 0.05$ 并用梯形方法求起动值. 图 6-23b 表示使用不同的两步方法. 它的不稳定性将是在 6.7.2 节中稳定性分析的主题.

一般的 s 步方法形如

$$w_{i+1} = a_1 w_i + a_2 w_{i-1} + \cdots + a_s w_{i-s+1} + h[b_0 f_{i+1} + b_1 f_i$$

$$+b_2f_{i-1}+\cdots+b_sf_{i-s+1}]. \tag{6.73}$$

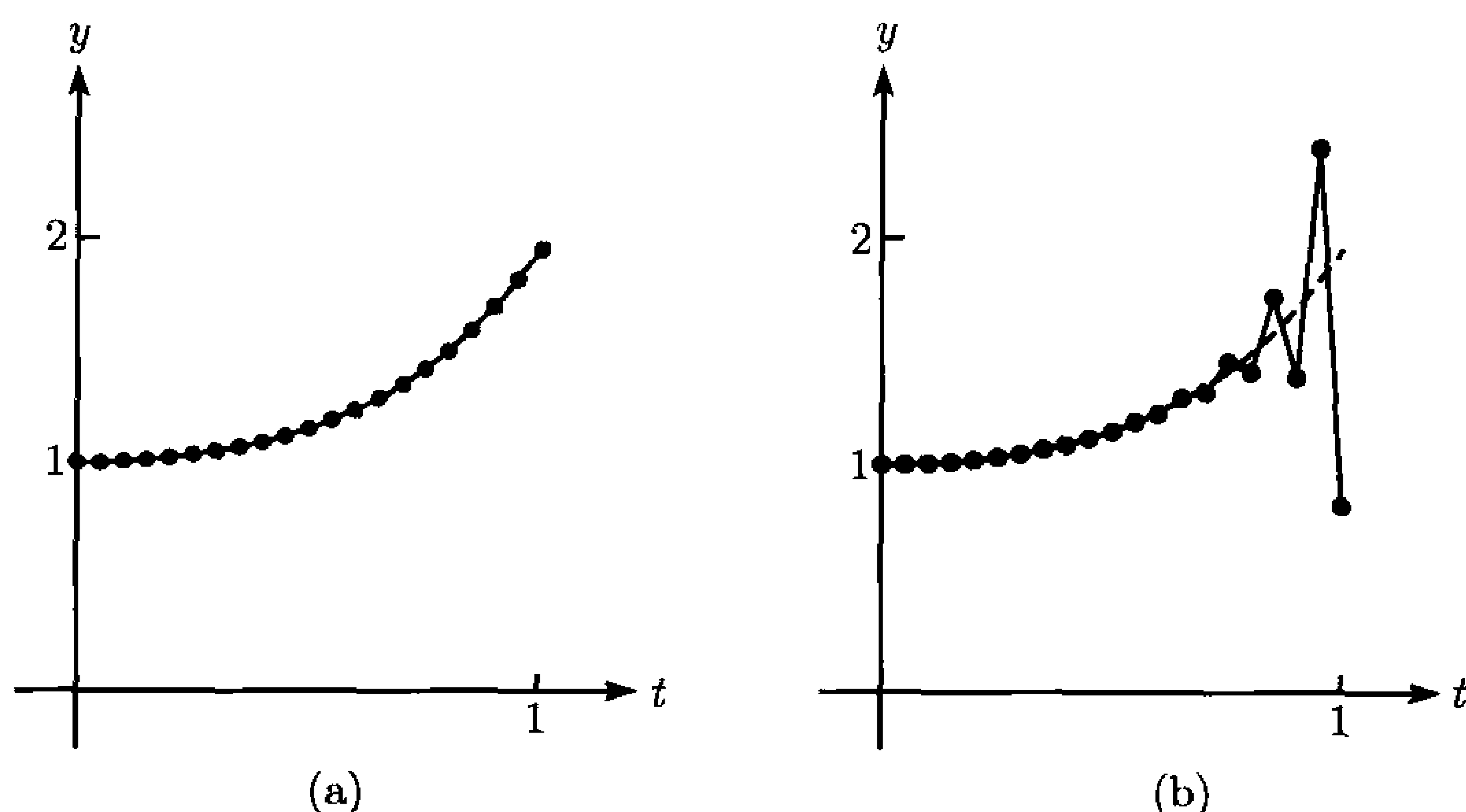


图 6-23 用于初值问题 (6.5) 的两步方法. 虚曲线表示准确解. 步长 $h=0.05$. (a) Adams-Bashforth 两步方法如圆圈所画. (b) 不稳定方法 (6.81) 在圆圈里

步长是 h, 而且我们用方便的记号

$$f_i\equiv f(t_i,w_i).$$

如果 $b_0=0$, 则方法是显式的. 假如 $b_0\neq 0$, 则方法是隐式的. 随后将讨论如何简单地使用隐式方法.

首先, 我们要说明多步方法是如何导出的, 以及如何决定哪一种将做得最好. 以相对简单的两步方法为例来介绍多步方法. 一般的两步方法 [在 (6.73) 中令 $s=2$] 有以下形式:

$$w_{i+1}=a_1w_i+a_2w_{i-1}+h[b_0f_{i+1}+b_1f_i+b_2f_{i-1}]. \tag{6.74}$$

为了建立多步方法, 我们参考 Taylor 定理, 因为手法仍是把解的 Taylor 展式的项尽可能多地与方法的项相匹配. 余下的部分将是局部截断误差.

假定所有前面的 w_i 是准确的, 即在 (6.74) 中 $w_i=y_i$ 及 $w_{i-1}=y_{i-1}$. 由微分方程知 $y_i'=f_i$, 所以所有的项都能如下 Taylor 展开:

$$\begin{aligned}w_{i+1}=&a_1w_i+a_2w_{i-1}+h[b_0f_{i+1}+b_1f_i+b_2f_{i-1}]\\=&a_1[y_i]+a_2[y_i-hy_i'+\frac{h^2}{2}y_i''-\frac{h^3}{6}y_i'''+\frac{h^4}{24}y_i''''-\cdots]\\&+b_0[hy_i'+h^2y_i''+\frac{h^3}{2}y_i'''+\frac{h^4}{6}y_i''''+\cdots]\\&+b_1[hy_i']+b_2[hy_i'-h^2y_i''+\frac{h^3}{2}y_i'''-\frac{h^4}{6}y_i''''+\cdots].\end{aligned}$$

相加得到

$$\begin{aligned} w_{i+1} = &(a_1 + a_2)y_i + (b_0 + b_1 + b_2 - a_2)hy_i' + (a_2 - 2b_2 + 2b_0)\frac{h^2}{2}y_i'' \\ &+(-a_2 + 3b_0 + 3b_2)\frac{h^3}{6}y_i''' + (a_2 + 4b_0 - 4b_2)\frac{h^4}{24}y_i'''' + \cdots . \end{aligned} \tag{6.75}$$

通过近似地选取 a_i 和 b_i, 可以使得局部截断误差 $y_{i+1} - w_{i+1}$ 尽可能小, 只要所包含的导数精确地存在, 这里

$$y_{i+1} = y_i + hy_i' + \frac{h^2}{2}y_i'' + \frac{h^3}{6}y_i''' + \cdots . \tag{6.76}$$

下面我们将研究可能性.

6.7.2 显式多步方法

为了得到显式方法, 令 $b_0 = 0$. 通过匹配 (6.75) 和 (6.76) 中的项直到包含 h^2 项, 使局部截断误差为 $O(h^3)$, 就能建立一个二阶方法. 比较同类项系数得到方程组

$$\begin{cases} a_1 + a_2 = 1, \\ -a_2 + b_1 + b_2 = 1, \\ a_2 - 2b_2 = 1. \end{cases} \tag{6.77}$$

这里有 4 个未知量和 3 个方程, 所以将可能求得无穷多种显式的二阶方法 (其中一个解相应于 3 阶方法, 见习题 3). 注意上面的方程可以用 a_1 来表示:

$$\begin{cases} a_2 = 1 - a_1, \\ b_1 = 2 - \frac{1}{2}a_1, \\ b_2 = -\frac{1}{2}a_1. \end{cases} \tag{6.78}$$

局部截断误差将是

$$\begin{aligned} y_{i+1} - w_{i+1} &= \frac{1}{6}h^3y_i''' - \frac{3b_2 - a_2}{6}h^3y_i''' + O(h^4) \\ &= \frac{1 - 3b_2 + a_2}{6}h^3y_i''' + O(h^4) = \frac{4 + a_1}{12}h^3y_i''' + O(h^4). \end{aligned} \tag{6.79}$$

亮点 复杂性

对单步方法来讲, 多步方法的优点是显然的. 在最初若干步之后, 只需要对右端函数作一次新的计算. 而对于单步方法, 一般需要几次函数计

算. 例如 4 阶 Runge-Kutta 方法每步需要计算 4 次函数值, 而 4 阶 Adams-Bashforth 方法在起动阶段后仅需要 1 次.

可以任意设置 a_1：任意一种选择都导出一种二阶方法, 如前所示. 设 $a_1 = 1$ 就得到二阶 Adams-Bashforth 方法 (6.72). 由第一个方程知 $a_2 = 0$, 因此 $b_2 = -\frac{1}{2}, b_1 = \frac{3}{2}$. 根据 (6.79), 局部截断误差是 $\frac{5}{12}h^3y'''(t_i) + O(h^4)$.

或者, 我们也可以设 $a_1 = \frac{1}{2}$, 这样就得到另外一种两步二阶方法, 这时 $a_2 = \frac{1}{2}, b_1 = \frac{7}{4}, b_2 = -\frac{1}{4}$:

$$w_{i+1} = \frac{1}{2}w_i + \frac{1}{2}w_{i-1} + h\left[\frac{7}{4}f_i - \frac{1}{4}f_{i-1}\right]. \tag{6.80}$$

这种方法有局部截断误差 $\frac{3}{8}h^3y'''(t_i) + O(h^4)$.

第三种选择,$a_1 = -1$, 给出下面这种两步二阶方法:

$$w_{i+1} = -w_i + 2w_{i-1} + h\left[\frac{5}{2}f_i + \frac{1}{2}f_{i-1}\right]. \tag{6.81}$$

它被用在图 6-23b 中. (6.81) 的失败引出一种重要的多步解法必须满足的稳定性条件. 考虑甚至更简单的初值问题

$$\begin{cases} y' = 0, \\ y(0) = 0, \\ t \in [0, 1]. \end{cases} \tag{6.82}$$

把方程 (6.81) 用到这个例子中, 得到

$$w_{i+1} = -w_i + 2w_{i-1} + h[0]. \tag{6.83}$$

(6.83) 的一个解是 $w_i \equiv 0$. 但是还有别的解. 把形式为 $w_i = c\lambda^i$ 的解代入 (6.83) 得

$$\begin{aligned} c\lambda^{i+1} + c\lambda^i - 2c\lambda^{i-1} &= 0, \\ c\lambda^{i-1}(\lambda^2 + \lambda - 2) &= 0. \end{aligned} \tag{6.84}$$

这个递推关系的 "特征多项式"$\lambda^2 + \lambda - 2 = 0$ 的根是 1 和 −2. 后者是个问题: 它意味着对某个常数 c, 形式为 $(-2)^ic$ 的解是这种方法的解. 这就使小的舍入和截断误差迅速增大到显著的大小并且淹没整个计算, 如图 6-23 所示. 为了避免这种可能性, 重要的是方法的特征多项式的根按绝对值不大于 1. 这就导出以下定义.

定义 6.6 *假如多项式 $P(x) = x^s - a_1x^{s-1} - \cdots - a_s$ 的根按绝对值不大于 1, 并且任何绝对值为 1 的根都是单根, 那么多步方法 (6.73) 就是***稳定***的. 1 是绝对值为 1 的唯一根的稳定方法称为***强稳定***, 否则就是***弱稳定***.

Adams-Bashforth 方法 (6.72) 有根 0 和 1, 它是强稳定的, 而方法 (6.81) 有根 -2 和 1, 它是不稳定的.

利用 (6.78) 中 $a_1 = 1 - a_2$ 的事实, 一般的两步公式的特征多项式是

$$P(x) = x^2 - a_1x - a_2 = x^2 - a_1x - 1 + a_1 = (x-1)(x - a_1 + 1),$$

它的根是 1 和 $a_1 - 1$. 回到 (6.78), 通过设 $a_1 = 0$, 我们可以得到一个弱稳定的二阶方法. 于是根是 1 和 -1, 导出以下弱稳定的两步二阶方法:

$$w_{i+1} = w_{i-1} + 2hf_i. \tag{6.85}$$

例 6.26 把强稳定方法 (6.72)、弱稳定方法 (6.85) 和不稳定方法 (6.81) 用到初值问题

$$\begin{cases} y' = -3y, \\ y(0) = 1, \\ t \in [0,2]. \end{cases} \tag{6.86}$$

其解为曲线 $y = \mathrm{e}^{-3t}$. 我们将用程序 6.7 来跟踪观察解, 其中`ydot.m`已改为

```
function z=ydot(t,y)
z=-3*y;
```

`ab2step`被 3 个调令`ab2step`, `weaklystabe2step`或`unstable2step`之一所代替.

图 6-24 显示了步长 $h = 0.1$ 时 3 个解的逼近. 弱稳定方法和不稳定方法似乎紧跟准确解一会儿, 然后很快地离开了它. 尽管减小步长, 可以延迟不稳定性的发作, 但是没有消除这个问题. ◀

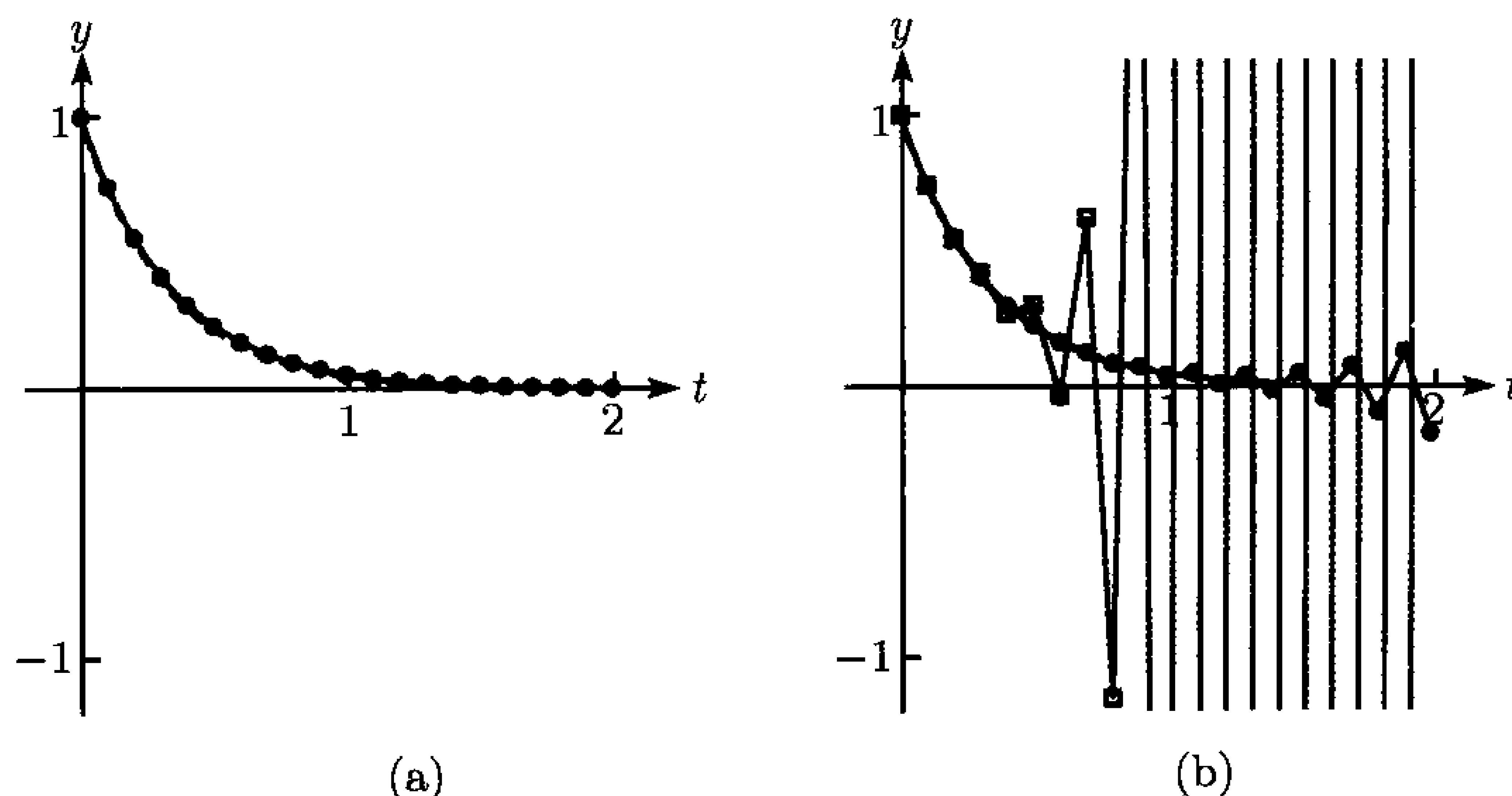


图 6-24 初值问题 (6.86) 的两步二阶方法的比较.

(a) Adams-Bashforth 方法;

(b) 弱稳定方法 (圆圈) 和不稳定方法 (正方形)

用下面两个定义可以叙述多步解法的基本定理.

定义 6.7 如果一种多步方法至少是一阶的, 那么它是**相容的**. 若当 $h \to 0$ 时, 对每一个 t, 其近似解收敛于准确解, 则这种解法是**收敛**的.

定理 6.8 (Dahlquist) 如果初始值是准确的, 那么多步方法 (6.73) 是收敛的当且仅当它是稳定的和相容的.

Dahlquist 定理的证明可参见 [12]. 定理 6.8 告诉我们, 避免像图 6-24b 中那样的两步二阶方法的灾难性失败, 就像检查方法的稳定性一样简单.

特征多项式的一个根必须是 1(见习题 6). Adams-Barhforth 方法是其余根都是零的那种. 因此, Adams-Bashforth 两步方法被认为是最稳定的两步方法.

使用更多步的高阶方法的推导恰好类似于我们前面两步方法的推导. 习题 13 和习题 14 要求证明下面的方法是强稳定的.

Adams-Bashforth 3 步方法(三阶)

$$w_{i+1} = w_i + \frac{h}{12}[23f_i - 16f_{i-1} + 5f_{i-2}]. \tag{6.87}$$

Adams-Bashforth 4 步方法(四阶)

$$w_{i+1} = w_i + \frac{h}{24}[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}]. \tag{6.88}$$

6.7.3 隐式多步方法

当 (6.73) 中的系数 b_0 不是零时, 方法是隐式的. 最简单的二阶隐式方法 (见习题 5) 是隐式梯形方法:

隐式梯形方法(二阶)

$$w_{i+1} = w_i + \frac{h}{2}[f_{i+1} + f_i]. \tag{6.89}$$

如果用 Euler 方法对 w_{i+1} 计算 f 在 "预测" 处的值来代替 f_{i+1}, 那么这就变成显式梯形方法. 这种隐式梯形方法也叫做 Adams-Moulton 一步方法. 这是由于隐式梯形方法和 Adams-Moulton 方法的步骤类似. 两步隐式方法的例子是 Adams-Moulton 两步方法.

Adams-Moulton 2 步方法(三阶)

$$w_{i+1} = w_i + \frac{h}{12}[5f_{i+1} + 8f_i - f_{i-1}]. \tag{6.90}$$

隐式方法和显式方法之间有重要的差别. 首先, 隐式情形不像显式情形, 可以

只用两步就得到一种稳定的 3 阶隐式方法. 其次, 对于隐式方法, 相应的局部截断误差公式比较小. 另一方面隐式方法有它固有的困难, 即必须计算隐式部分的值这种额外的处理.

由于这些原因, 隐式方法常常在 “预测-校正” 对中用于校正. 同阶的隐式和显式方法一起使用. 每一步都是显式方法的预测和隐式方法的校正的结合, 这里隐式方法用预测的 w_{i+1} 来计算 f_{i+1}. 由于计算微分方程的右端 f 是既用这一步的预测又用校正来完成的, 所以预测-校正方法用了两次近似计算的努力. 然而增加的精确性和稳定性常常使得这种努力很值得.

一种简单的预测-校正方法把 Adams-Bashforth 两步显式方法作为预测与把 Adams-Moulton 单步隐式方法作为校正结成一对. MATLAB 代码看似与以前用过的 Adams-Barhforth 代码类似, 但是加上一校正步.

```
% Program 6.8 Adams-Bashforth-Moulton second-order p-c
% Inputs: [inter(1),inter(2)] time interval
% ic=[y0]  initial condition
% h=stepsize, s=number of steps for explicit method
% Calls multistep methods such as ab2step.m and am1step.m
% Example usage: predcorr([0 1],1,.05,2)
function [t,y]=predcorr(inter,ic,h,s)
n=round((inter(2)-inter(1))/h);
% Start-up phase
y(1,:)=ic;t(1)=inter(1);
for i=1:s-1                 % start-up phase, using one-step method
  t(i+1)=t(i)+h;
  y(i+1,:)=trapstep(t(i),y(i,:),h);
  f(i,:)=ydot(t(i),y(i,:));
end
for i=s:n                   % multistep method loop
  t(i+1)=t(i)+h;
  f(i,:)=ydot(t(i),y(i,:));
  y(i+1,:)=ab2step(t(i),i,y,f,h);  % predict
  f(i+1,:)=ydot(t(i+1),y(i+1,:));
  y(i+1,:)=am1step(t(i),i,y,f,h);  % correct
end

function y=trapstep(t,x,h)
%one step of the Trapezoid Method from section 6.2
z1=ydot(t,x);
g=x+h*z1;
z2=ydot(t+h,g);
y=x+h*(z1+z2)/2;

function z=ab2step(t,i,y,f,h)
%one step of the Adams-Bashforth 2-step method
```

```
z=y(i,:)+h*(3*f(i,:)-f(i-1,:))/2;
function z=am1step(t,i,y,f,h)
%one step of the Adams-Moulton 1-step method
z=y(i,:)+h*(f(i+1,:)+f(i,:))/2;

function z=ydot(t,y)  % IVP
z=t*y+t^3;
```

Adams-Moulton 两步方法的推导恰如显式方法的建立. 重新解方程组 (6.77), 但不需要 $b_0 = 0$. 因为现在有一个额外的参数 (b_0), 我们可以只用一个两步方法通过 3 次项把 (6.75) 和 (6.76) 匹配起来, 把局部截断误差放到 h^4 项中. 与 (6.77) 类似的是

$$\begin{aligned} a_1 + a_2 &= 1, \\ -a_2 + b_0 + b_1 + b_2 &= 1, \\ a_2 + 2b_0 - 2b_2 &= 1, \\ -a_2 + 3b_0 + 3b_2 &= 1. \end{aligned} \tag{6.91}$$

满足这些方程的结果便是一个 3 阶两步隐式方法.

以上方程能用 a_1 表示, 形式如下:

$$\begin{aligned} a_2 &= 1 - a_1, \qquad b_0 = \frac{1}{3} + \frac{1}{12}a_1, \\ b_1 &= \frac{4}{3} - \frac{2}{3}a_1, \quad b_2 = \frac{1}{3} - \frac{5}{12}a_1. \end{aligned} \tag{6.92}$$

局部截断误差是

$$\begin{aligned} y_{i+1} - w_{i+1} &= \frac{1}{24}h^4 y_i'''' - \frac{4b_0 - 4b_2 + a_2}{24}h^4 y_i'''' + O(h^5) \\ &= \frac{1 - a_2 - 4b_0 + 4b_2}{24}h^4 y_i'''' + O(h^5) \\ &= -\frac{a_1}{24}h^4 y_i'''' + O(h^5). \end{aligned}$$

只要 $a_1 \neq 0$, 这个方法就是 3 阶. 因为 a_1 是自由参数, 所以有无穷多种 3 阶两步隐式方法. Adams-Moulton 两步方法使用的选择是 $a_1 = 1$. 习题 8 要求证明这种方法是强稳定的. 习题 9 探究了 a_1 的其他选择.

注意一种更特别的选取,$a_1 = 0$. 从局部截断误差公式可知, 这个两步方法是 4 阶的.

Milne-Simpson 方法

$$w_{i+1} = w_{i-1} + \frac{h}{3}[f_{i+1} + 4f_i + f_{i-1}]. \tag{6.93}$$

习题 10 要求你验证它仅是弱稳定的. 因此, 它对误差放大很敏感.

隐式梯形方法 (6.89) 和 Milne-Simpson 方法 (6.93) 中提到的术语, 应使读者想起第 5 章中的数值积分公式. 事实上, 尽管我们没有强调这种逼近, 但我们提出的许多多步公式可以用另一种方法通过积分近似插值式而得到, 这类似于一种数值积分方案.

隐藏在这种逼近后面的思想是, 微分方程 $y' = f(t, y)$ 可以在区间 $[t_i, t_{i+1}]$ 上积分得到

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(t, y)\mathrm{d}t. \tag{6.94}$$

用一种数值积分方案去逼近 (6.94) 中的积分, 结果就得到一种多步常微分方程方法. 例如用第 5 章中的梯形法则进行数值积分就得到

$$y(t_{i+1}) - y(t_i) = \frac{h}{2}(f_{i+1} + f_i) + O(h^2),$$

这是常微分方程的二阶梯形方法. 假如用 Simpson 法则进行近似积分, 结果是

$$y(t_{i+1}) - y(t_i) = \frac{h}{3}(f_{i+1} + 4f_i + f_{i-1}) + O(h^4),$$

这是 4 阶 Milne-Simpson 方法 (6.93). 本质上, 我们通过多项式和积分来逼近常微分方程的右端, 恰如在数值积分中做的那样. 只要通过改变插值多项式的次数和插值点的位置就能推广这种逼近, 以覆盖许多我们已经提出的多步方法. 尽管这种逼近是导出多步方法的更几何的方法, 但是它没能使我们对所得出的常微分方程解法的稳定性有特别的了解.

通过推广前面的方法, 在每种情况中用 $a_1 = 1$, 就能推导出高阶 Adams-Moulton 方法.

Adams-Moulton 三步方法(四阶)

$$w_{i+1} = w_i + \frac{h}{24}[9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}]. \tag{6.95}$$

Adams-Moulton 四步方法(五阶)

$$w_{i+1} = w_i + \frac{h}{720}\left[251f_{i+1} + 646f_i - 264f_{i-1} + 106f_{i-2} - 19f_{i-3}\right]. \tag{6.96}$$

这些方法以及具有同阶的 Adams-Bashforth 预测的方法, 大量地用在预测–校正方法中. 计算机问题 5 和问题 6 用 MATLAB代码实现这种思想.

习题 6.7

1. 对以下初值问题用 Adams-Barhforth 方法.

(a) $y'=t$; (b) $y'=t^2y$; (c) $y'=2(t+1)y$; (d) $y'=5t^4y$; (e) $y'=\frac{1}{y^2}$;
(f) $y'=\frac{t^3}{y^2}$.
初始条件是 $y(0)=1$, 在区间 $[0,1]$ 上用步长 $h=\frac{1}{4}$. 用显式梯形方法产生 w_1, 利用在习题 6.1.3 中的准确解, 求出在 $t=1$ 处的整体截断误差.

2. 对下列初值问题执行习题 1 中的步骤.
   (a) $y'=t+y$; (b) $y'=t-y$; (c) $y'=4t-2y$.
   初始条件是 $y(0)=0$. 利用习题 6.1.4 中的准确解, 求出在 $t=1$ 处的整体截断误差.
3. 求一个两步三阶的显式方法. 这个方法是否稳定?
4. 求一个特征多项式有重根 1 的两步二阶显式方法.
5. 证明隐式梯形方法 (6.89) 是二阶方法.
6. 对于 $s\geqslant 2$, 如果显式或隐式的 s 步方法至少是一阶的, 则其特征多项式一定有一根是 1, 为什么?
7. (a) 对什么样的 a_1, 存在强稳定两步二阶显式方法?
   (b) 对弱稳定方法, 回答同样的问题.
8. 证明 Adams-Moulton 两步方法的系数满足 (6.92), 因此这种方法是强稳定的.
9. 对下列两步隐式方法求出它们的阶和稳定的类型:
   (a) $w_{i+1}=3w_i-2w_{i-1}+\frac{h}{12}[13f_{i+1}-20f_i-5f_{i-1}]$;
   (b) $w_{i+1}=\frac{4}{3}w_i-\frac{1}{3}w_{i-1}+\frac{2}{3}hf_{i+1}$;
   (c) $w_{i+1}=\frac{4}{3}w_i-\frac{1}{3}w_{i-1}+\frac{h}{9}[4f_{i+1}+4f_i-2f_{i-1}]$;
   (d) $w_{i+1}=3w_i-2w_{i-1}+\dfrac{h}{12}[7f_{i+1}-8f_i-11f_{i-1}]$;
   (e) $w_{i+1}=2w_i-w_{i-1}+\frac{h}{2}[f_{i+1}-f_{i-1}]$.
10. 从 (6.92) 中导出 Milne-Simpson 方法 (6.93), 并证明它是四阶弱稳定的.
11. 求一个弱稳定的两步二阶隐式方法.
12. Milne-Simpson 方法是一种弱稳定的两步四阶隐式方法. 有没有弱稳定两步三阶隐式方法?
13. (a) 对一个三步三阶显式方法, 求 a_i,b_i 要求的条件 (类似于 (6.77)). (b)证明 Adams-Bashforth 三步方法满足这些条件. (c) 证明 Adams-Barhforth 三步方法是强稳定的. (d)求一个弱稳定的三步三阶显式方法, 并证明这些性质.
14. (a) 对一个四步四阶显式方法, 求 a_i,b_i 要求的条件 (类似于 (6.77)). (b) 证明 Adams-Barhforth 四步方法满足这些条件. (c) 证明 Adams-Barhforth 四步方法是强稳定的.
15. (a) 对一个三步四阶隐式方法, 求 a_i,b_i 要求的条件 (类似于 (6.77)). (b) 证明 Adams-Moulton 三步方法满足这些条件. (c) 证明 Adams-Moulton 三步方法是强稳定的.

计算机问题 6.7

1. 修改`exmultistep.m`程序, 对习题中的初值问题用 Adams-Barforth 两步方法. 取步长 $h=0.1$, 计算在区间 $[0,1]$ 上的近似解. 打印出 t 值、近似解以及每步的整体截断误差的表格.
2. 修改`exmultistep.m`程序, 对习题 2 中的初值问题用 Adams-Barhforth 两步方法. 取 $h=0.1$, 计算在区间$[0,1]$上的近似解. 打印出t 值、近似解和每步的整体截断误差的表格.

3. 使用不稳定的两步方法 (6.81), 执行计算机问题 2 中的步骤.
4. 使用 Adams-Barhforth 三步方法, 执行计算机问题 2 中的步骤.
5. 把程序 6.8 改变成三阶预测–校正方法, 用 Adams-Bashforth 三步方法和 Adams-Moulton 两步方法, 取步长 $h = 0.05$. 在区间 $[0, 5]$ 画出初值问题 (6.5) 的近似解和准确解.
6. 把程序 6.8 改变为三阶预测–校正方法, 用 Adams-Bashforth 四步方法和 Adams-Moulton 三步方法, 取步长 $h = 0.05$. 画出在区间 $[0, 5]$ 上的初值问题 (6.5) 的近似解和准确解.

软件和进一步阅读

关于常微分基础的传统出处是 [4,5,6,10,16]. 许多书采用大量的计算与绘图来讲授微分方程的基础, ODE Architect[8] 是一个好的例子. Polking 的 MATLAB 代码和指南 [19] 是学习和使常微分方程概念形象化的极好方法.

为了补充求解常微分方程中单步和多步的数值方法, 还有许多中级的和高级的教科书. 教科书 [14,11] 是经典的, 从 [21] 得到一种当代的 MATLAB逼近. 其他推荐的教科书是 [15,20,1,17, 9,7] 以及综合性的两卷集 [12,13].

有许多复杂的软件可用来解常微分方程. MATLAB使用的解法的细节能在 [22,2] 中找到. Runge-Kutta 型变步长显式方法通常对非刚性或轻度刚性问题是成功的. 除了 Runge-Kutta-Fehlberg 和 Dormand-Prince 之外, 一种 5/6 阶 Runge-Kutta-Verner 方法也经常使用. 对于刚性问题, 需要使用后向差分方法和外推方法.

IMSL 程序库包括有基于 Runge-Kutta-Verner 方法的双精度程序 DIVPRK, 以及能解决刚性问题的多步 Adams 类方法的程序 DIVPAG. NAG 程序库提供了可运行标准 Runge-Kutta 步骤的驱动程序 D02BJF. 包含了带有误差控制的 Adams 型程序的多步驱动是 D02CJF. 对于刚性问题, 推荐使用程序 D02EJF, 其中用户可有选择地指定使用 Jacobi 方法以加速计算.

Netlib 程序库包括 Runge-Kutta-Fehlberg 方法的 Fortran 程序 RKF45, 及 Runge-Kutta-Verner 方法的 DVERK. Netlib 软件包 ODE 包含几种多步 (方法) 程序. 程序 VODE 可用于解决刚性问题.

软件包 ODEPACR 是由 Lawrence Livermore 国家实验室 (LLNL) 发展起来的求解 ODE 问题的 Fortran 程序的公共区域集, 其基本解法器 LSODE 及其变体可用于解决刚性和非刚性问题. 这些程序可在 LLNL 的网站 http://www.llnl.gov/CASC/odepack 上免费获得.

第7章 边值问题

地下和海底管线必须能够承受外部环境的压力. 管子越深, 防止破裂所花的代价越大. 连结北海平台到海岸的石油管线埋在 70m 深处. 天然气重要性日益凸显, 而船运既危险又昂贵, 这些都可能导致建设国际气管线. 大西洋中部深逾 5km, 那里 7 000 psi 的水压将要求管材的革新和避免弯曲的构造.

管道的弯曲理论对从建筑支撑到管状伸展的广泛应用是重要的. 当直接实验太昂贵太困难时, 弯曲的数值模拟是很有价值的.

实例检验 7.1.2 节后的实例检验是关于作为圆环的管道的横截面, 检查什么时候和如何出现弯曲.

第 6 章叙述了计算初值问题 (IVP) 解的方法, 初值问题是微分方程及其给定在求解区间左端的初始条件的一个组合. 我们提出的方法都是 "前进式" 技巧–近似解开始于左端并且关于自变量 t 向前进展. 当我们把微分方程连同求解区间两端给定的边界条件一起给出时, 就产生了一类同样重要的问题.

第 7 章叙述边值问题 (BVP) 的近似解法. 方法有 3 种类型. 首先, 提出打靶法, 这是一种把第 6 章里的 IVP 解法和第 1 章里方程的解法结合在一起的方法. 其次, 采用有限差分方法, 它把微分方程及其边界条件化为待解的线性或非线性方程组. 最后一节重点讨论配置法和 Galerkin 方法, 它们通过把解表示为元素基函数来求解这个问题.

7.1 打靶法

第一种方法通过确定与边界条件相容的缺失初值把边值问题转化为初值问题. 把第 1 章和第 6 章中已经建立起来的方法结合在一起就能做到这一点.

7.1.1 边值问题的解

通常二阶边值问题要求在区间 $a \leqslant t \leqslant b$ 上解

$$\begin{cases} y'' = f(t, y, y'), \\ y(a) = y_a, \\ y(b) = y_b, \end{cases} \tag{7.1}$$

如图 7-1 所示. 如第 6 章所述, 在典型的光滑性条件下, 微分方程有无穷多个解, 需要额外的条件去确定一个特解. 在 (7.1) 中, 方程是二阶的, 需要两个约束条件, 给定为解 $y(t)$ 在 a 处和 b 处的边界条件.

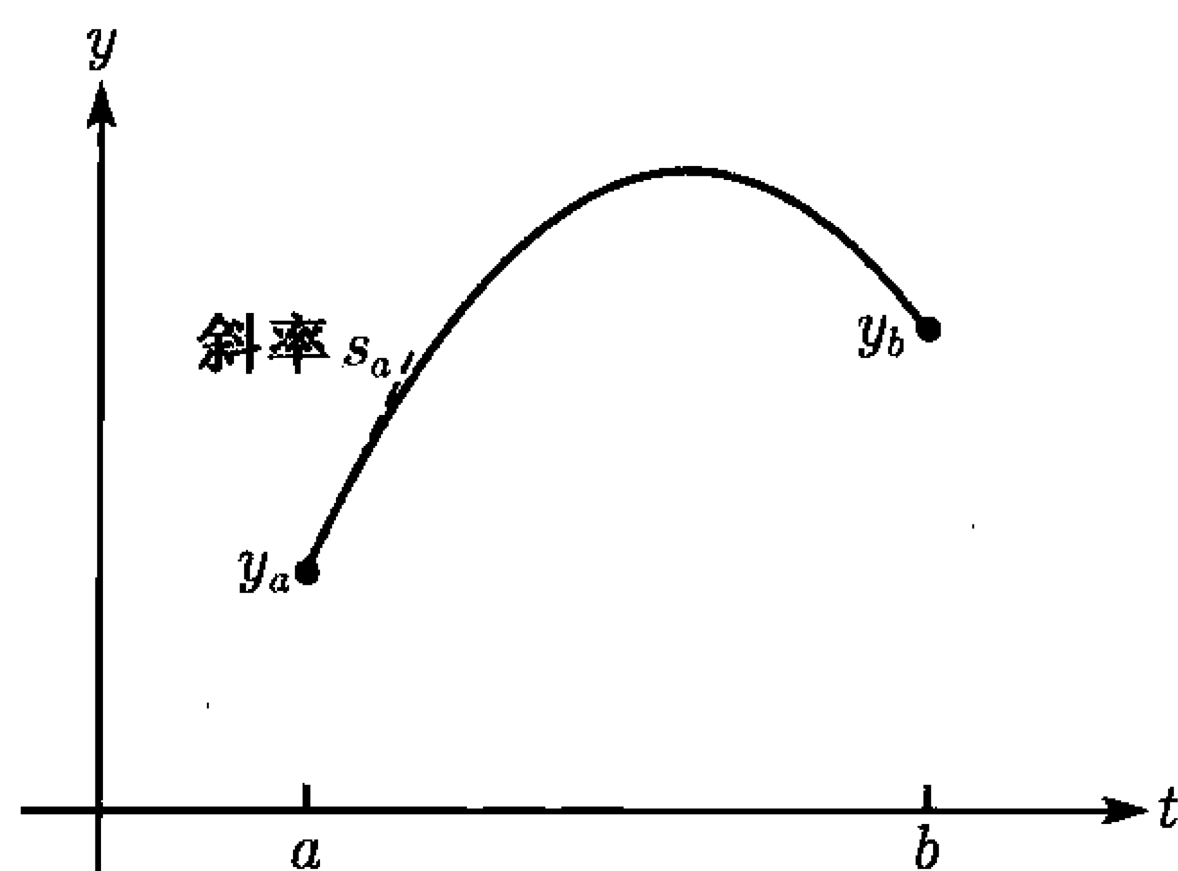


图 7-1 IVP 和 BVP 的比较: 在初值问题中, 给定初值 y_a 和初始梯度 $s_a = y'(a)$ 作为问题的一部分; 在边值问题中, 给定边值 y_a 和 y_b, 而 s_a 未知

为了更加直观, 考虑一个抛射体 (或一发炮弹), 当它运动时, 满足二阶微分方程 $y''(t) = -g$, 这里 y 是抛射体的高度而 g 是重力加速度. 作为初值问题, 确定了初始位移和初始速度就唯一确定了抛射体的运动. 另一方面, 可以指定一个时间区间以及位移 $y(a)$ 和 $y(b)$. 后一个问题, 即边值问题, 在这种情形下也有唯一解.

例 7.1 一发炮弹在 120 英尺高的大楼顶层开始发射, 3 秒钟后着地, 求它的弹道 (轨迹).

根据 Newton 第二定律 $F = ma$, 可导出微分方程, 这里重力是

$$F = -mg, \qquad g = 32 \text{ 英尺/秒}^2.$$

设 $y(t)$ 是炮弹在 t 时的高度, 弹道 (轨迹) 能表示为以下 IVP 的解:

$$\begin{cases} y'' = -g, \\ y(0) = 120, \\ y'(0) = v_0, \end{cases}$$

或者是以下 BVP 的解:

$$\begin{cases} y'' = -g, \\ y(0) = 120, \\ y(3) = 0. \end{cases}$$

因为我们并不知道初始速度 v_0, 所以必须解边值问题. 两次积分得到

$$y(t) = -\frac{1}{2}gt^2 + v_0t + y_0 = -16t^2 + v_0t + y_0.$$

利用边值条件得到

$$120 = y(0) = y_0,$$
$$0 = y(3) = -144 + 3v_0 + 120,$$

这就意味着 $v_0 = 8$ 英尺/秒. 解轨迹就是 $y(t) = -16t^2 + 8t + 120$. ◀

例 7.2 证明 $y(t) = t\sin t$ 是以下二阶微分方程的解:

$$y'' = -y + 2\cos t, \tag{7.2}$$

其边界条件是

$$y(0) = 0,$$
$$y(\pi) = 0.$$

函数 $y(t) = t\sin t$ 如图 7-2 所示, 这个函数是微分方程的解, 这是因为

$$y''(t) = (2\cos t - t\sin t) = -t\sin t + 2\cos t.$$

检验边界条件给出 $y(0) = 0\sin(0) = 0$ 及 $y(\pi) = \pi\sin\pi = 0$. ◀

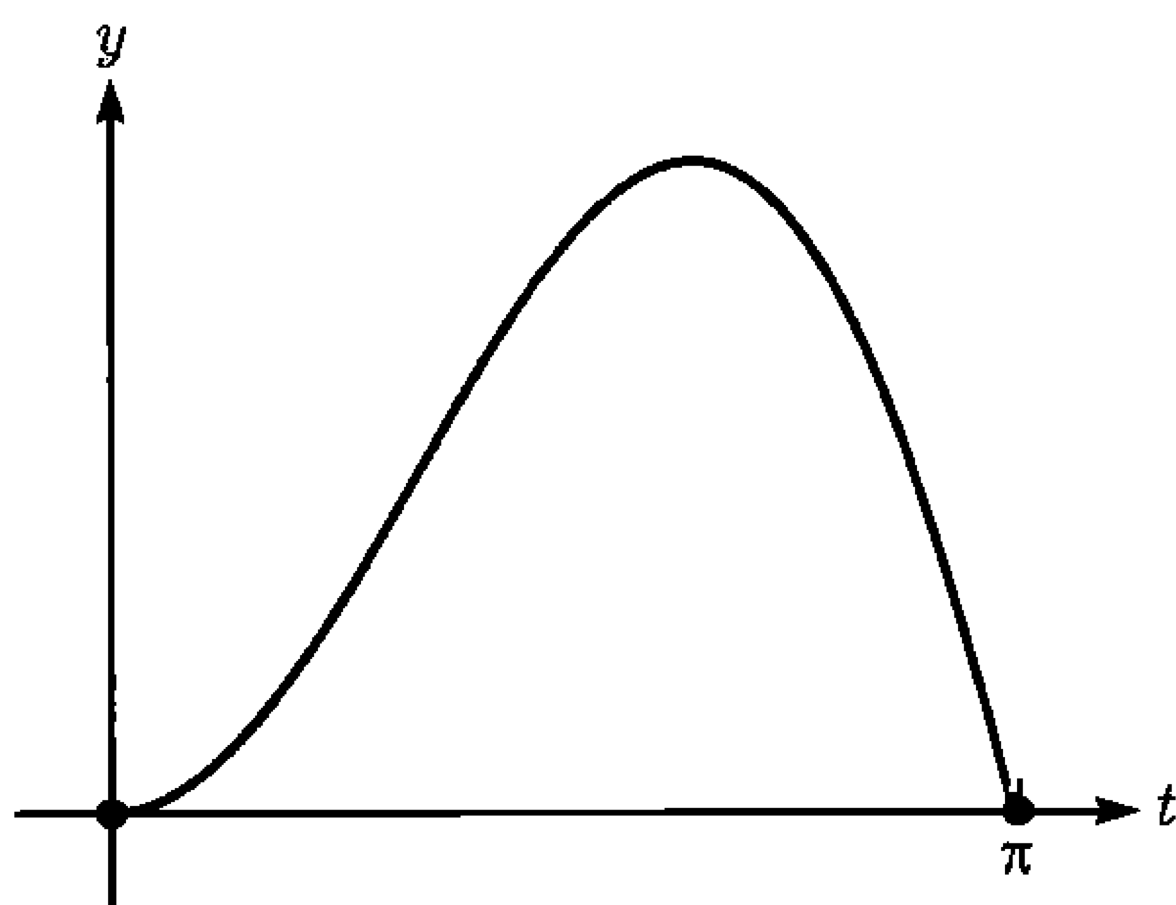


图 7-2 BVP(7.2) 的解: $y(t) = t\sin t$ 带有边界值 $y(0) = 0, y(\pi) = 0$ 的解的图示

边值问题解的存在性和唯一性理论远比初值问题相应的理论复杂得多. 表面上合理的边值问题可能无解或有无穷多解, 这不同于初值问题.

这种存在性和唯一性的情形类似于炮弹在地球引力作用下运动的弧线. 假定炮弹有固定的离开火炮的初速度, 但是火炮角度可变化, 任一原来位置和速度的数值将决定由于地球引力而产生的弹道轨迹. 初值问题的唯一解永远存在. 边值问题有不同的性质. 假如执行者的网格设置在火炮的范围之外, 解就不存在. 而且, 对于在火炮范围内的任何边界条件, 存在两个解,"短程"(火炮点火角度小于 45°) 及 "长

程"(角度大于 45°), 这就违反了唯一性. 下面两例对于非常简单的微分方程说明了这两种可能性.

例 7.3 证明边值问题

$$\begin{cases} y'' = -y, \\ y(0) = 0, \\ y(\pi) = 1 \end{cases}$$

无解.

微分方程有一个二维族解. 它是由两个线性独立的解 $\cos t$ 和 $\sin t$ 产生的. 方程的所有解一定具有形式 $y(t) = a\cos t + b\sin t$. 代入第一个初始条件 $0 = y(0) = a$, 意味着 $a = 0$, 即 $y(t) = b\sin t$. 第二个边界条件 $1 = y(\pi) = b\sin \pi = 0$ 得出矛盾. 因此不存在解, 存在性不成立. ◀

例 7.4 证明边值问题

$$\begin{cases} y'' = -y, \\ y(0) = 0, \\ y(\pi) = 0 \end{cases}$$

有无穷多解.

可以检验, 对每一个实数 $k, y(t) = k\sin t$ 是微分方程的解并且满足边界条件. 特别地, 对这个例子而言, 解的唯一性不成立. ◀

例 7.5 求以下边值问题的全部解:

$$\begin{cases} y'' = 4y, \\ y(0) = 1, \\ y(1) = 3. \end{cases} \tag{7.3}$$

这个例题简单得足以精确求解, 还很有兴趣, 可以作为边值问题解法所遵循的例子. 我们能猜出微分方程的两个解, $y = \mathrm{e}^{2t}$ 和 $y = \mathrm{e}^{-2t}$. 因为这两个解彼此不成倍数, 它们是线性独立的, 所以从微分方程的基本理论知道, 微分方程的全部解是线性组合 $c_1\mathrm{e}^{2t} + c_2\mathrm{e}^{-2t}$. 两个常数 c_1 和 c_2 由以下两个边界条件确定:

$$\begin{aligned} 1 &= y(0) = c_1 + c_2, \\ 3 &= y(1) = c_1\mathrm{e}^2 + c_2\mathrm{e}^{-2}. \end{aligned}$$

解出这两个常数得到解:

$$y(t) = \frac{3 - \mathrm{e}^{-2}}{\mathrm{e}^2 - \mathrm{e}^{-2}}\mathrm{e}^{2t} + \frac{\mathrm{e}^2 - 3}{\mathrm{e}^2 - \mathrm{e}^{-2}}\mathrm{e}^{-2t}. \tag{7.4}$$

◀

7.1.2　打靶法的实现

打靶法通过应用一系列初值问题来解边值问题 (7.1). 首先, 随着初值 y_a 给出初始梯度 s_a. 按这个初始梯度解出这个初值问题并与边界值 y_b 进行比较. 经过试验和尝试, 改进初始梯度直到与边界值相匹配. 为了对这种方法给出一种更规范的结构, 定义函数

$$F(s)=\begin{cases} y_b\text{与}y(b)\text{的差,} \\ \text{这里}y(t)\text{是满足}y(a)=y_a\text{及} \\ y'(a)=s\text{的初值问题的解.} \end{cases}$$

根据这个定义, 边值问题就化为解方程

$$F(s)=0, \tag{7.5}$$

如图 7-3 所示.

我们现在可以用第 1 章中的方程解法来解这个方程. 假定选择对分法. 必须找到 s 的两个值, 设为 s_0 和 s_1, 使得 $F(s_0)F(s_1)<0$. 然后由 s_0 和 s_1 括住了 (7.5) 的一个根. 令 $s_2=(s_0+s_1)/2$, 并且把这种对分进行下去直到求得在允许误差内的解 s^*. 于是这个解就被当作初值问题

$$\begin{cases} y''=f(t,y,y'), \\ y(a)=y_a, \\ y'(a)=s^* \end{cases} \tag{7.6}$$

的解而求出来 (例如用第 6 章中的初值问题的解法).

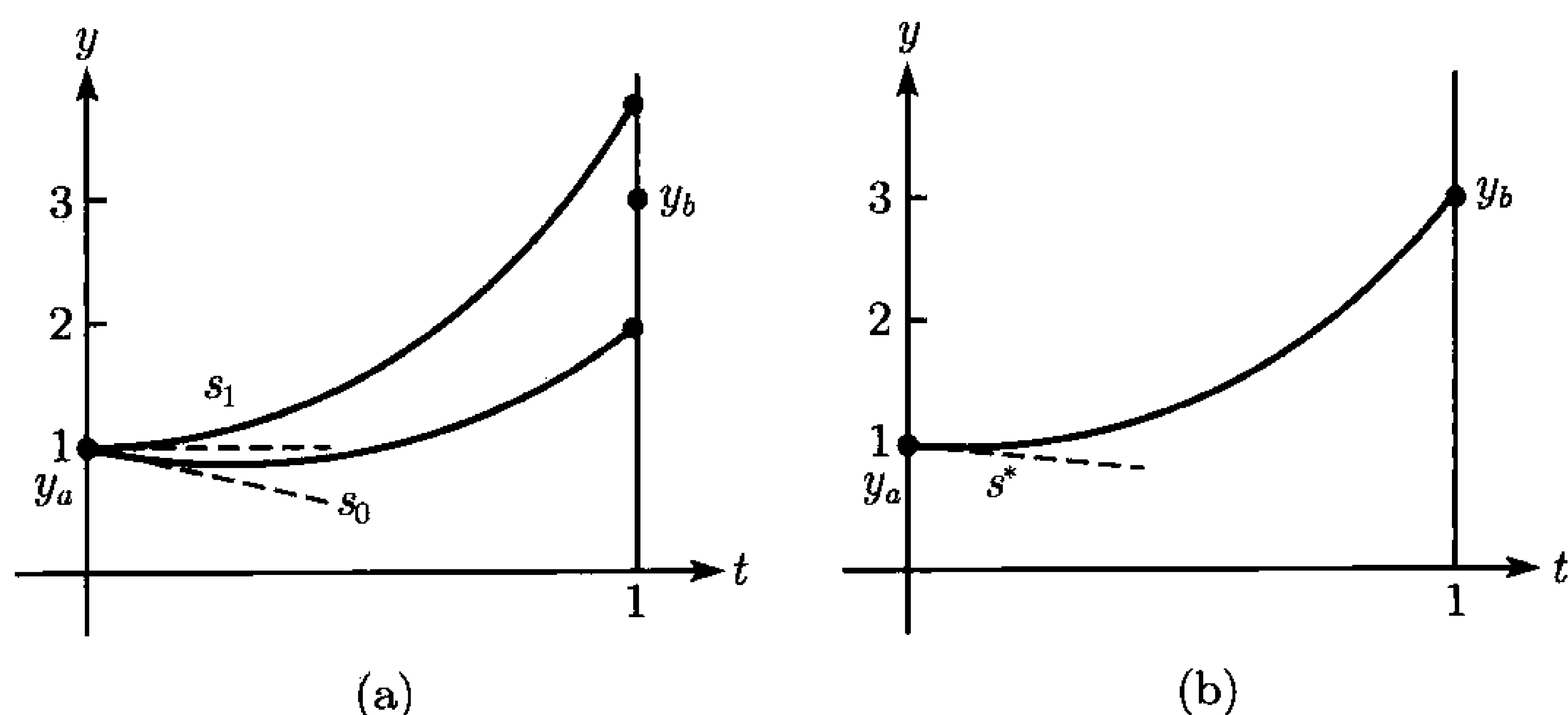


图 7-3　打靶法. (a) 解边值问题, 对于初始估计 s_0, 解初始条件为 $y(a)=y_a, y'(a)=s_0$ 的初值问题. $F(s_0)$ 的值是 $y(b)-y_b$. 然后选取新的 s_1, 并且在求解 $F(s)=0$ 的目标下重复此过程, 以求出 s. (b) 对根 s^*, 用 MATLAB的ode45绘制初值问题 (7.7) 的解

例 7.6 用打靶法解边值问题

$$\begin{cases} y'' = 4y, \\ y(0) = 1, \\ y(1) = 3. \end{cases} \tag{7.7}$$

为了使用 MATLAB的ode45 IVP 解法, 把微分方程写为一阶方程组

$$\begin{cases} y' = v, \\ v' = 4y. \end{cases} \tag{7.8}$$

写出函数文件de.m作为对ode45的输入:

```
function ydot=f(y)
ydot=[0;0];
ydot(1)=y(2);
ydot(2)=4*y(1);
```

写出函数文件F.m 作为第 1 章中bisect.m的输入:

```
function z=F(s)
a=0; b=1; yb=3;
[t,y]=ode45('f',[a,b],[0,s])
z=y(end,1)-yb; % end means last entry of solution y
```

算出 $F(-1) \approx -1.05, F(0) \approx 0.76$, 这能在图 7-3a 中观察到. 因此在 -1 和 0 之间有 F 的一个根, 运行第 1 章中的bisect.m, 从区间 $[-1, 0]$ 出发, 在所要求的精度内求出 s, 然后这个解能被当作一个初值问题的解 (见图 7-3b) 绘制出来. (7.7) 的精确解由 (7.4) 给出, 因此 $s^* = y'(0) \approx -0.420\,3$. ◀

对于常微分方程组, 边值问题有多种形式. 下面探究一种可能的形式并引导读者参考习题和实例检验 7 作为进一步的例子.

例 7.7 用打靶法解边值问题:

$$\begin{cases} y_1' = \dfrac{4 - 2y_2}{t^3}, \\ y_2' = -\mathrm{e}^{y_1}, \\ y_1(1) = 0, \\ y_2(2) = 0, \\ t \in [1, 2]. \end{cases} \tag{7.9}$$

假如给出初始条件 $y_2(1)$, 这将是一个初值问题. 我们用打靶法确定出未知量 $y_2(1)$, 调用 MATLAB程序ode45解初值问题. 定义函数 $F(s)$ 为端点条件 $y_2(2)$, 这里初值问题在初始条件 $y_1(1) = 0$ 及 $y_2(1) = s$ 下已解. 目标就是求解 $F(s) = 0$.

注意到 $F(0) \approx -3.97$ 及 $F(2) \approx 0.87$, 因此 $F(s)=0$ 的解被括在 0 和 2 之间. 对 $F(s)$ 应用割线法, 从初始估计 $s_0=0, s_1=2$ 出发, 大概 6 步, 收敛到精确到若干位小数的 $s^*=1.5$. 使用带有初值 $y_1(1)=0$ 及 $y_2(1)=1.5$ 的ode45进行求解, 结果描绘在图 7-4 中. 精确解是 $y_1(t)=\ln t,\ y_2(t)=2-\frac{t^2}{2}$. ◀

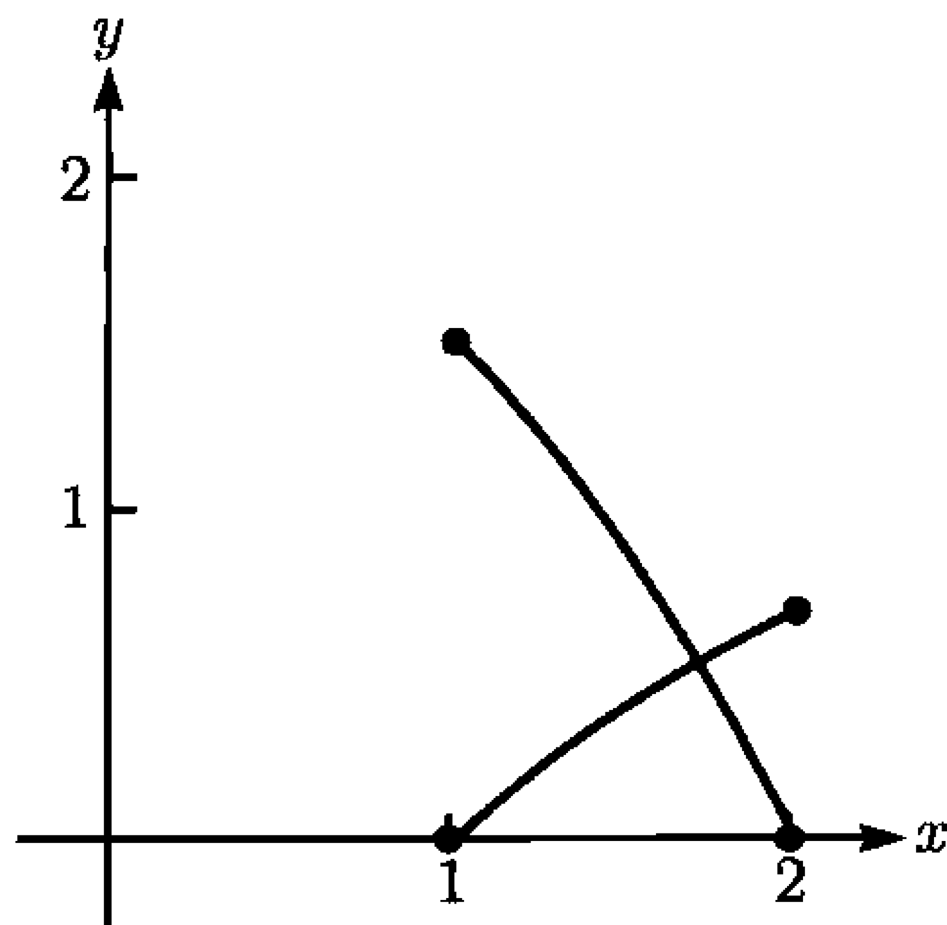


图 7-4 打靶法求出的例 7.7 的解. 显示曲线 $y_1(t)$ 和 $y_2(t)$. 实心圆点表示已给出的边界条件

习题 7.1

1. 对于线性边值问题

(a) $\begin{cases} y''=y+2\mathrm{e}^t, \\ y(0)=0, \\ y(1)=\mathrm{e}; \end{cases}$ (b) $\begin{cases} y''=(2+4t^2)y, \\ y(0)=1, \\ y(1)=\mathrm{e}; \end{cases}$

(c) $\begin{cases} y''=-y+2\cos t, \\ y(0)=0, \\ y\left(\frac{\pi}{2}\right)=\frac{\pi}{2}; \end{cases}$ (d) $\begin{cases} y''=2-4y, \\ y(0)=0, \\ y\left(\frac{\pi}{2}\right)=1. \end{cases}$

证明它们的解分别是 (a) $y=t\mathrm{e}^t$; (b) $y=\mathrm{e}^{t^2}$; (c) $y=t\sin t$; (d) $y=\sin^2 t$.

2. 对于边值问题

(a) $\begin{cases} y''=\frac{3}{2}y^2, \\ y(1)=4, \\ y(2)=1; \end{cases}$ (b) $\begin{cases} y''=2yy', \\ y(0)=0, \\ y\left(\frac{\pi}{4}\right)=1; \end{cases}$ (c) $\begin{cases} y''=-\mathrm{e}^{-2y}, \\ y(1)=0, \\ y(\mathrm{e})=1; \end{cases}$ (d) $\begin{cases} y''=6y^{\frac{1}{3}}, \\ y(1)=1, \\ y(2)=8. \end{cases}$

证明它们的解分别是 (a)$y=4t^{-2}$; (b) $y=\tan t$; (c) $y=\ln t$; (d) $y=t^3$.

3. 考虑边值问题

$$\begin{cases} y''=-4y, \\ y(a)=y_a, \\ y(b)=y_b. \end{cases}$$

(a)求该微分方程的两个线性独立的解. **(b)** 设 $a=0, b=\pi$, 为使解存在, y_a, y_b 必须满足什么条件? **(c)** 对 $b=\frac{\pi}{2}$ 回答 (b) 中的问题. **(d)** 对 $b=\frac{\pi}{4}$, 回答 (b) 中的问题.

4. 从 200 英尺高的大楼扔出的抛射体, 5 秒钟后着地, 把它的高度表示成二阶边值问题的解. 然后解这个边值问题并且求这个抛射体的最大高度.

5. 求边值问题 $y''=ky, y(0)=y_0, y(1)=y_1 (k \geqslant 0)$ 的解.

计算机问题 7.1

1. 对下面线性边值问题用打靶法. 通过寻找能括住解的区间 $[s_0, s_1]$ 开始, 用对分法求解. 在指定的区间绘制近似解的图.

(a) $\begin{cases} y'' = y + \frac{2}{3}\mathrm{e}^t, \\ y(0) = 0, \\ y(1) = \frac{1}{3}\mathrm{e}; \end{cases}$ (b) $\begin{cases} y'' = (2 + 4t^2)y, \\ y(0) = 1, \\ y(1) = \mathrm{e}. \end{cases}$

2. 对下列边值问题执行计算机问题 1 的步骤:

(a) $\begin{cases} 9y'' + \pi^2 y = 0, \\ y(0) = -1, \\ y\left(\frac{3}{2}\right) = 3; \end{cases}$ (b) $\begin{cases} y'' = 3y - 2y', \\ y(0) = \mathrm{e}^3, \\ y(1) = 1. \end{cases}$

3. 用打靶法解非线性边值问题, 求包含解的区间 $[s_0, s_1]$, 用割线法求解并绘制解的图.

(a) $\begin{cases} y'' = 18y^2, \\ y(1) = \frac{1}{3}, \\ y(2) = \frac{1}{12}; \end{cases}$ (b) $\begin{cases} y'' = 2e^{-2y}(1 - t^2), \\ y(0) = 0, \\ y(1) = \ln 2. \end{cases}$

4. 对以下非线性边值问题执行计算机问题 3 的步骤.

(a) $\begin{cases} y'' = \mathrm{e}^y, \\ y(0) = 1, \\ y(1) = 3; \end{cases}$ (b) $\begin{cases} y'' = \sin y', \\ y(0) = 1, \\ y(1) = -1. \end{cases}$

5. 仿效例 7.7 的方法, 对下面非线性方程组边值问题应用打靶法.

(a) $\begin{cases} y_1' = \frac{1}{y_2}, \\ y_2' = t + \tan y_1, \\ y_1(0) = 0, \\ y_2(1) = 2; \end{cases}$ (b) $\begin{cases} y_1' = y_1 - 3y_1 y_2, \\ y_2' = -6(ty_2 + \ln y_1), \\ y_1(0) = 1, \\ y_2(1) = -\dfrac{2}{3}. \end{cases}$

实例检验 7 圆环的挠度弯曲

边值问题是结构计算的自然模型. 7 个微分方程的方程组作为在来自任何方向的液压 p 下, 压缩系数为 c 的圆环的模型. 为了简单, 我们将对模型无量纲化, 并假定环的半径为 1, 并且在无外界压力的情形下具有水平与垂直的对称性. 这个模型尽管简单, 但它对**弯曲现象** (或者圆环形状压偏) 的研究是很有用的. 这个例子和许多其他结构边值问题能在 [4] 中找到.

这个模型仅考虑**环的左上四分之一**部分, 其余部分可根据对称性假设来填补. 自变量 s 表示环的弧长, 它沿着过圆环中心的直线从 $s = 0$ 到 $s = \frac{\pi}{2}$. 在由弧长 s 所确定的点处的因变量如下:

$y_1(s) =$ 过圆环中心的直线与水平方向的夹角

$y_2(s) = x$ 坐标

$y_3(s) = y$ 坐标

$y_4(s) =$ 沿变形中心线的弧长

$y_5(s) =$ 内轴向力

$y_6(s) =$ 内法向力

$y_7(s) =$ 弯矩

图 7-5a 展示了环以及前 4 个变量. 边值问题 (例如, 见 [4]) 是

$$
\begin{aligned}
&y_1' = -1 - cy_5 + (c+1)y_7, && y_1(0) = \frac{\pi}{2}, && y_1\left(\frac{\pi}{2}\right) = 0;\\
&y_2' = (1 + c(y_5 - y_7))\cos y_1, && && y_2\left(\frac{\pi}{2}\right) = 0;\\
&y_3' = (1 + c(y_5 - y_7))\sin y_1, && y_3(0) = 0;\\
&y_4' = 1 + c(y_5 - y_7), && y_4(0) = 0;\\
&y_5' = -y_6(-1 - cy_5 + (c+1)y_7),\\
&y_6' = y_7y_5 - (1 + c(y_5 - y_7))(y_5 + p), && y_6(0) = 0, && y_6\left(\frac{\pi}{2}\right) = 0;\\
&y_7' = (1 + c(y_5 - y_7))y_6.
\end{aligned}
$$

在没有压力 ($p = 0$) 时, 注意到 $y_1 = \frac{\pi}{2} - s, (y_2, y_3) = (-\cos s, \sin s), y_4 = s, y_5 = y_6 = y_7 = 0$ 是解. 这个解是一个完整的四分之一圆, 它对应具有对称性的完好的圆环.

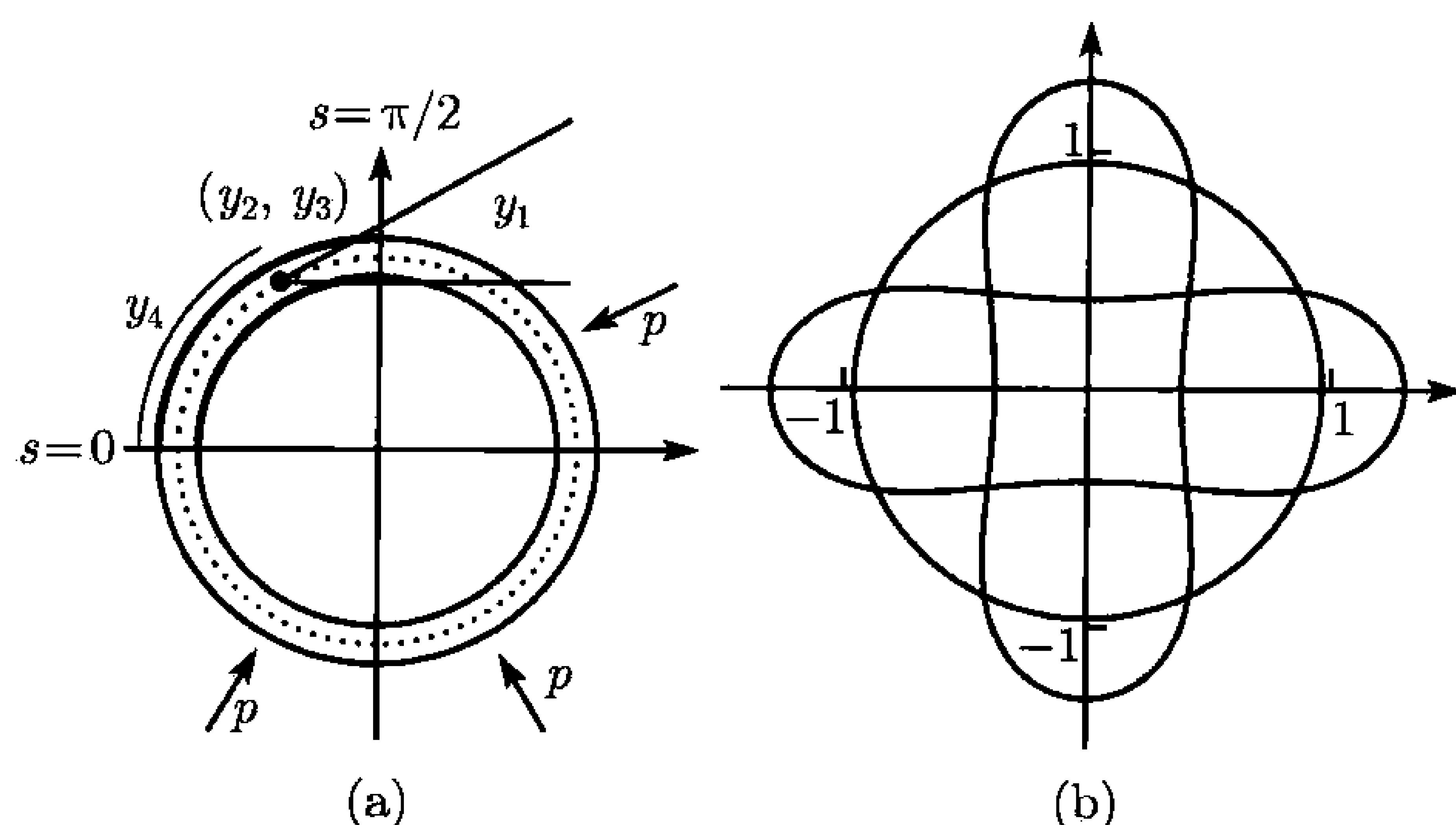


图 7-5 弯曲圆环示意图: (a) 变量 s 表示沿着环的左上方四分之一部分虚线中心线的弧长; (b) 参数 $c = 0.01, p = 3.8$ 时边值问题的 3 个不同的解, 两个弯曲的解是稳定的

事实上, 对任选的参数 c 和 p, 边值问题以下形式的圆环解是存在的:

$$
\begin{aligned}
y_1(s) &= \frac{\pi}{2} - s,\\
y_2(s) &= \frac{c+1}{cp+c+1}(-\cos s),\\
y_3(s) &= \frac{c+1}{cp+c+1}\sin s,\\
y_4(s) &= \frac{c+1}{cp+c+1}s,\\
y_5(s) &= -\frac{c+1}{cp+c+1}p,\\
y_6(s) &= 0,\\
y_7(s) &= -\frac{cp}{cp+c+1}.
\end{aligned}
\tag{7.10}
$$

当压力从零增大, 圆的半径就减小. 当压力参数 p 进一步增大, 就有了环的**分叉**或者可能状态的改变. 环可能在数学上保持圆环的形状, 但不稳定, 这就意味着小扰动使环移动到另一种可能形状 (边值问题的解), 而这种形状是稳定的.

当施加的压力 p 低于分叉点或**临界压力**p_c 时, 仅存在解 (7.10). 当 $p > p_c$ 时, 边值问题存在 3 个不同的解, 它们展现在图 7-5b 中. 超出临界压力时, 圆环作为不稳定状态的角色类似于倒的摆锤 (见计算机问题 6.3.6) 或在实例检验 6 中的无扭转的桥梁.

临界压强依赖于环的压缩系数. 参数 c 越小, 环的可压缩性越小, 使其变形的临界压力越低, 而不是原来形状的压缩. 你的任务是用打靶法及 Broyden 方法求出临界压力 p_c 以及由环得到的弯曲形状的结果.

建议习题

1. 证明 (7.10) 是边值问题对每一个压缩系数 c 以及压力 p 的解.
2. 置压缩系数为适中的值 $c = 0.01$. 对压力 $p = 0$ 及 $p = 3$ 用打靶法解边值问题, 打靶法中的函数 F 应该用到 3 个缺失的初值 $(y_2(0), y_5(0), y_7(0))$ 作为输入, 使用 3 个终值 $(y_1(\frac{\pi}{2}), y_2(\frac{\pi}{2}), y_6(\frac{\pi}{2}))$ 作为输出. 第 2 章中的多变量 Broyden II 解法可以用来求 F 的根. 与准确解 (7.10) 比较. 注意, 对于 p 的这两个值, Broyden 方法对不同的初始条件都给出相同的解轨迹. 当 p 从 0 增加到 3 时, 半径减小多少?
3. 绘出第 2 步中的解. 曲线 $(y_2(s), y_3(s))$ 表示环的左上方四分之一部分. 利用水平和垂直方向的对称性绘出整个环.
4. 把压力变为 $p = 3.5$, 求解边值问题. 注意得到的解依赖于 Broyden 方法使用的初始条件. 绘出每一个求得的解.
5. 对压缩系数 $c = 0.01$, 求临界压力 p_c, 精确到两位小数. 对于 $p > p_c$, 存在 3 个不同的解; 对于 $p < p_c$, 只有一个解 (7.10).
6. 对于减小的压缩系数 $c = 0.001$, 执行第 5 步的任务. 现在环是更易损坏. 在压缩系数减小的情况下, p_c 的改变是否与你的直觉一致?
7. 对于增加的压缩系数 $c = 0.05$, 执行第 5 步的任务.

7.2 有限差分方法

有限差分方法的基本思想是用离散的近似公式来代替微分方程中的导数, 并求解在网格上建立的方程组. 这种离散微分方程的方法也将用于第 8 章中的偏微分方程.

7.2.1 线性边值问题

设 $y(t)$ 是一个至少 4 阶连续可微的函数. 在第 5 章, 我们建立了一阶导数的离散近似公式

$$y'(t) = \frac{y(t+h) - y(t-h)}{2h} - \frac{h^2}{6}y'''(c) \tag{7.11}$$

以及二阶导数的离散近似公式

$$y''(t) = \frac{y(t+h) - 2y(t) + y(t-h)}{h^2} + \frac{h^2}{12}f^4(c). \tag{7.12}$$

它们都精确到误差为 $O(h^2)$ 阶.

有限差分方法包括用离散形式来代替微分方程中的导数, 并求解结果为较简单的代数方程. 见图 7-6. 把边界条件代到需要它们的方程组中.

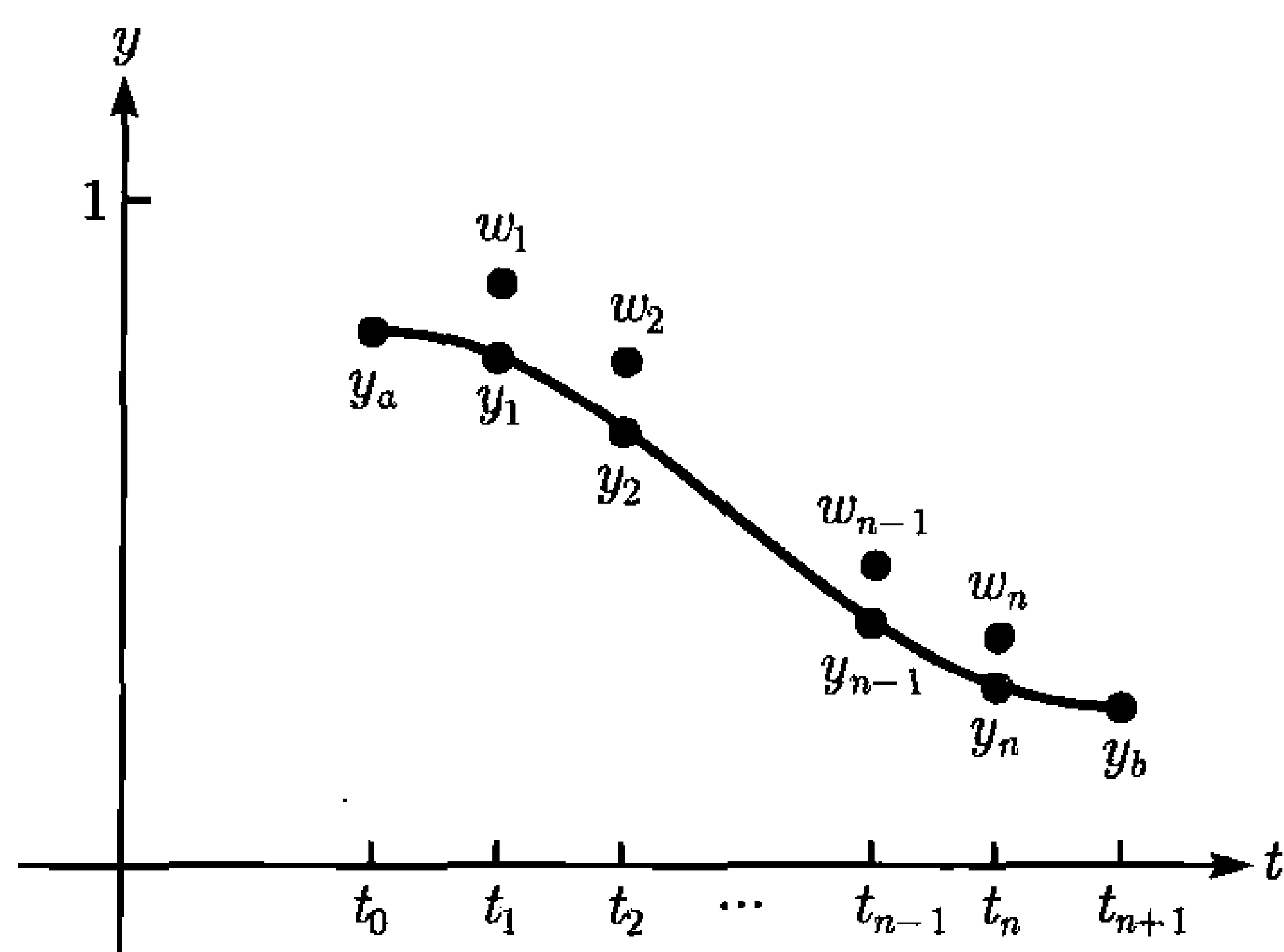


图 7-6 边值问题的有限差分方法: 通过解线性方程组, 计算在离散点 t_j 处准确值 y_i 的近似值 $w_i, i = 1, \cdots, n$

代入后可能有两种情形. 如果原边值问题是线性的, 那么得到的方程组是线性的, 可以用 Gaussian 消去法求解. 如果原问题是非线性的, 那么代数方程组是非线性方程组, 需要更复杂的方法求解. 我们从线性的例子开始.

例 7.8 用有限差分方法解边值问题 (7.7):

$$\begin{cases} y'' = 4y, \\ y(0) = 1, \\ y(1) = 3. \end{cases}$$

对二阶导数用中心差分形式, 考虑微分方程 $y'' = 4y$ 的离散形式. 在 t_i 处的有限差分形式是

$$\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} - 4w_i = 0,$$

或者等价地

$$w_{i-1} + (-4h^2 - 2)w_i + w_{i+1} = 0.$$

取 $n = 3$, 区间长 $h = \frac{1}{n+1} = \frac{1}{4}$, 这时有 3 个方程. 代入边界条件 $w_0 = 1$ 和 $w_4 = 3$, 需要求解以下方程组中的 w_1, w_2, w_3:

$$\begin{cases} 1 + (-4h^2 - 2)w_1 + w_2 = 0, \\ w_1 + (-4h^2 - 2)w_2 + w_3 = 0, \\ w_2 + (-4h^2 - 2)w_3 + 3 = 0. \end{cases}$$

代入 $h=\frac{1}{4}$ 便产生三对角矩阵方程:

$$\begin{bmatrix} -\frac{9}{4} & 1 & 0 \\ 1 & -\frac{9}{4} & 1 \\ 0 & 1 & -\frac{9}{4} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ -3 \end{bmatrix}.$$

用 Gaussian 消去法求解在 3 个点上的近似解的值是 1.024 9, 1.306 1, 1.913 8. 表 7-1 展示了 t_i 处解的近似值 w_i 与准确解的值 y_i 的比较 (注意边界值 w_0 和 w_4 事先已经知道而不再计算). 误差是 $O(10^{-2})$ 阶. 为了得到更小的误差, 需要用较大的 n. 一般地, $h=\frac{b-a}{n+1}=\frac{1}{n+1}$, 并且三对角矩阵方程是

$$\begin{bmatrix} -4h^2-2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -4h^2-2 & \ddots & & 0 & 0 & 0 \\ 0 & 1 & \ddots & \ddots & 0 & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & \ddots & \ddots & 1 & 0 \\ 0 & 0 & 0 & & \ddots & -4h^2-2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -4h^2-2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -3 \end{bmatrix}.$$

当加入更多的子区间时, 我们预期近似值 w_i 更接近相应的 y_i. ◀

表 7-1

i	t_i	w_i	y_i
0	0.00	1.000 0	1.000 0
1	0.25	1.024 9	1.018 1
2	0.50	1.306 1	1.296 1
3	0.75	1.913 8	1.904 9
4	1.00	3.000 0	3.000 0

有限差分方法误差的可能来源是中心差分公式产生的截断误差以及在解方程组中产生的误差. 对于大于机器 ε 平方根的步长 h, 前一个误差占主导地位, 这个误差是 $O(h^2)$, 因此, 我们预期当子区间的数目 $n+1$ 变大时, 误差随着 $O(n^{-2})$ 阶减小.

亮点 收敛

图 7-7 说明有限差分方法的二阶收敛性. 这是由于使用了二阶公式 (7.11) 和 (7.12). 阶的知识促使应用外推. 对于固定的 t 和步长 h, 用有限

差分方法得到的近似解 $w_h(t)$ 关于 h 是二阶的, 可以用简单的公式进行外推. 计算机问题 7 和问题 8 采用这种可能来加速收敛.

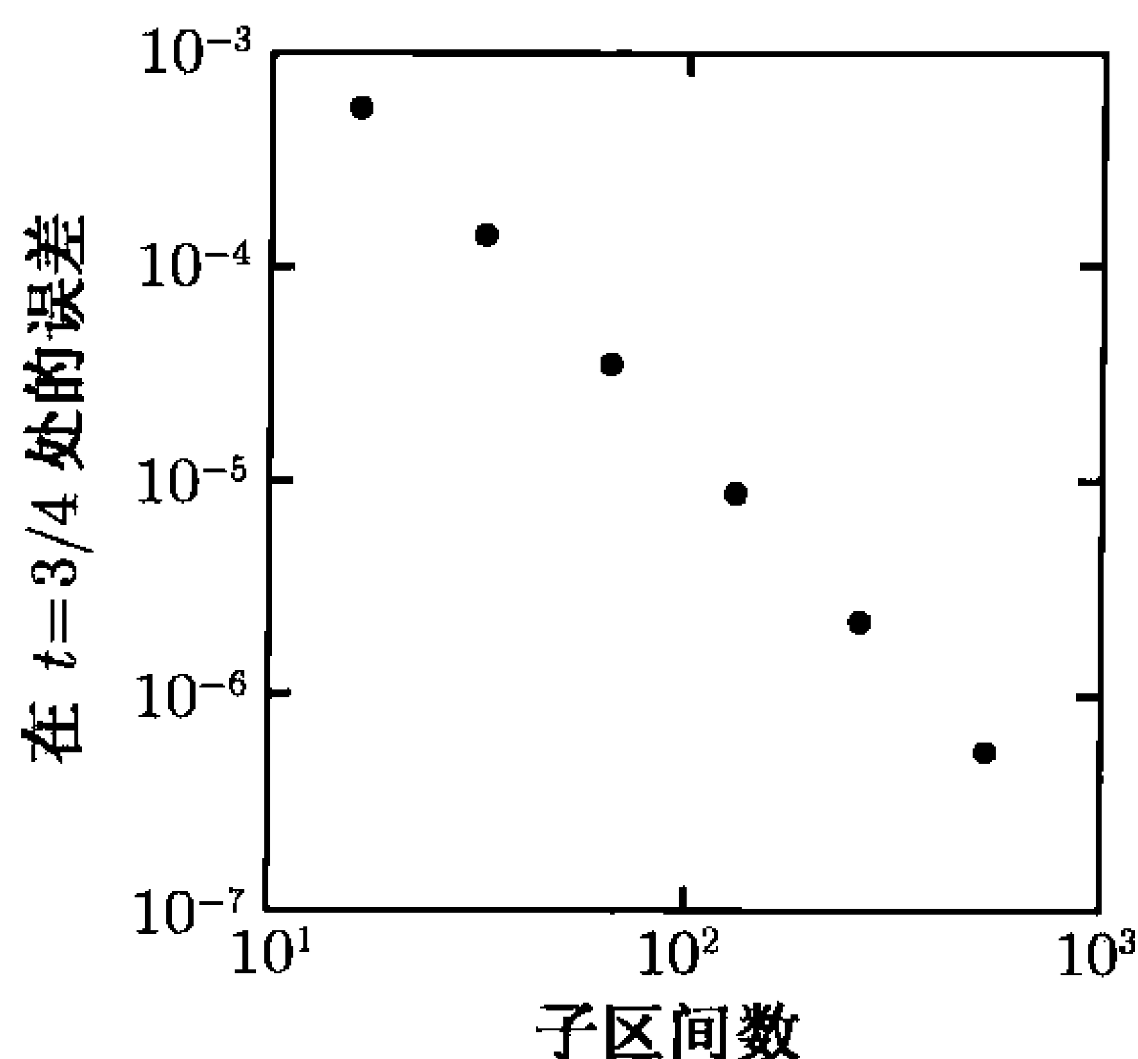


图 7-7 有限差分方法的收敛性. 例 7.8 中在 $t_i = \frac{3}{4}$ 处的误差作为子区间的数目 n 的函数用图表示. 斜率是 -2, 证实误差是 $O(n^{-2}) = O(h^2)$

我们就问题 (7.7) 测试上述收敛性. 对不同的子区间数 $n+1$, 图 7-7 表明了在 $t = \frac{3}{4}$ 处解的误差的大小. 在双对数坐标图上, 作为子区间数函数的误差实质上是一条斜率为 -2 的直线, 这意味着 $\ln E \equiv a + b\ln n$, 其中 $b = -2$; 换言之, 正如我们所料的, 误差 $E \approx Kn^{-2}$.

7.2.2 非线性边值问题

一个非线性微分方程应用有限差分方法时, 结果是要求解一个非线性代数方程组. 在第 2 章, 我们用多变量 Newton 法来解这样的方程组. 我们用 Newton 法来说明如何求以下非线性边值问题的近似解.

例 7.9 用有限差分方法解非线性边值问题

$$\begin{cases} y'' = y - y^2, \\ y(0) = 1, \\ y(1) = 4. \end{cases} \tag{7.13}$$

微分方程在 $t_i (2 \leqslant i \leqslant n-1)$ 处的离散形式是

$$\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} - w_i + w_i^2 = 0$$

或者

$$w_{i-1} - (2+h^2)w_i + h^2 w_i^2 + w_{i+1} = 0,$$

第一个方程和最后一个方程是

$$y_a - (2+h^2)w_1 + h^2 w_1^2 + w_2 = 0,$$
$$w_{n-1} - (2+h^2)w_n + h^2 w_n^2 + y_b = 0,$$

这两个方程携带了边界条件信息.

解边值问题的离散形式意味着解 $\boldsymbol{F}(\boldsymbol{w}) = \mathbf{0}$. 我们用 Newton 法来求解 $F(w) = 0$. 多变量 Newton 法就是迭代 $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \mathrm{D}\boldsymbol{F}(\boldsymbol{w}^k)^{-1}\boldsymbol{F}(\boldsymbol{w}^k)$. 通常, 最好是通过求解方程 $\mathrm{D}\boldsymbol{F}(\boldsymbol{w}^k)\Delta\boldsymbol{w} = -\boldsymbol{F}(\boldsymbol{w}^k)$ 中的 $\Delta\boldsymbol{w} = \boldsymbol{w}^{k+1} - \boldsymbol{w}^k$ 来进行迭代. 函数 $\boldsymbol{F}(\boldsymbol{w})$ 由下式给出:

$$\boldsymbol{F}\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix} = \begin{bmatrix} y_a - (2+h^2)w_1 + h^2 w_1^2 + w_2 \\ w_1 - (2+h^2)w_2 + h^2 w_2^2 + w_3 \\ \vdots \\ w_{n-2} - (2+h^2)w_{n-1} + h^2 w_{n-1}^2 + w_n \\ w_{n-1} - (2+h^2)w_n + h^2 w_n^2 + y_b \end{bmatrix},$$

这里 $y_a = 1, y_b = 4$. $\boldsymbol{F}$ 的 Jacobi 矩阵 $\mathrm{D}\boldsymbol{F}(\boldsymbol{w})$ 是

$$\begin{bmatrix} 2h^2w_1 - (2+h^2) & 1 & 0 & \cdots & 0 \\ 1 & 2h^2w_2 - (2+h^2) & \ddots & \ddots & \vdots \\ 0 & 1 & \ddots & 1 & 0 \\ \vdots & \ddots & \ddots & 2h^2w_{n-1} - (2+h^2) & 1 \\ 0 & \cdots & 0 & 1 & 2h^2w_n - (2+h^2) \end{bmatrix}.$$

Jacobi 矩阵的第 i 行是通过对第 i 个方程 (F 的第 i 分量) 关于每个 w_j 求偏导数而确定的.

图 7-8a 显示了用多变量 Newton 法解 $\boldsymbol{F}(\boldsymbol{w}) = \mathbf{0}$ (对 $n = 40$) 的结果. MATLAB 代码在程序 7.1 中给出. Newton 法进行 20 步就足以达到机器精度的收敛性.

```
% Program 7.1 Nonlinear Finite Difference Method for BVP
% Uses Multivariate Newton's Method to solve nonlinear equation
% Inputs: interval inter, boundary values bv, number of steps n
% Output: solution w
% Example usage: W=nlbvpfd([0 1], [1 4], 40)
function w=nlbvpfd(inter,bv,n);
global h n ya yb                  % needed in f and jac functions
a=inter(1); b=inter(2); ya=bv(1); yb=bv(2)
h=(b-a)/(n+1);                    % h is step size
w=zeros(n,1);                     % initialize solution array w
for i=1:20                        % loop of Newton step
```

```
  w=w-jac(w)\f(w);
end
plot([a (1:n)*h b],[ya w' yb]);  % plot w with boundary data

function y=f(w)
global h n ya yb
y=zeros(n,1);
y(1)=ya-(2+h^2)*w(1)+h^2*w(1)^2+w(2);
y(n)=w(n-1)-(2+h^2)*w(n)+h^2*w(n)^2+yb;
for i=2:n-1
    y(i)=w(i-1)-(2+h^2)*w(i)+h^2*w(i)^2+w(i+1);
end

function a=jac(w)
global h n ya yb
a=zeros(n,n);
for i=1:n
  a(i,i)=2*h^2*w(i)-2-h^2;
end
for i=1:n-1
  a(i,i+1)=1;
  a(i+1,i)=1;
end
```

◀

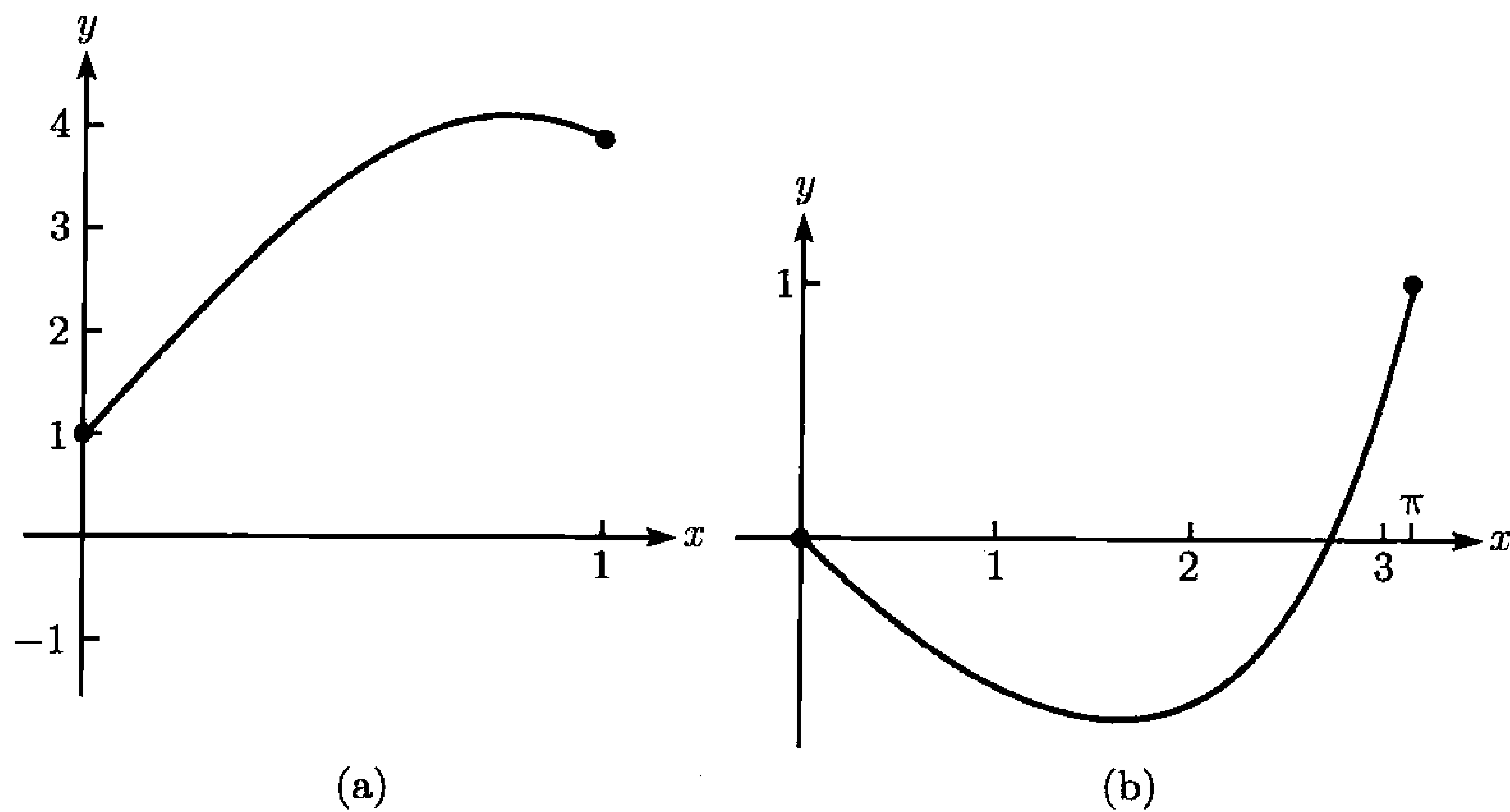


图 7-8 用有限差分法求解非线性边值问题：(a) 当 $n = 40$ 时，例 7.9 的 Newton 法收敛后的解；(b) 对例 7.10 同上

例 7.10 用有限差分法解非线性边值问题：

$$\begin{cases} y'' = y' + \cos y, \\ y(0) = 0, \\ y(\pi) = 1. \end{cases} \tag{7.14}$$

微分方程在 $t_i(2 \leqslant i \leqslant n-1)$ 处的离散形式是

$$\frac{w_{i+1}-2w_i+w_{i-1}}{h^2}-\frac{w_{i+1}-w_{i-1}}{2h}-\cos(w_i)=0,$$

或者

$$\left(1+\frac{h}{2}\right)w_{i-1}-2w_i+\left(1-\frac{h}{2}\right)w_{i+1}-h^2\cos w_i=0,$$

第一个方程和最后一个方程分别是

$$\left(1+\frac{h}{2}\right)y_a-2w_1+\left(1-\frac{h}{2}\right)w_2-h^2\cos w_1=0,$$
$$\left(1+\frac{h}{2}\right)w_{n-1}-2w_n+\left(1-\frac{h}{2}\right)y_b-h^2\cos w_n=0,$$

这里 $y_a=0, y_b=1$. 上述 n 个方程的左端形成一个向量值函数

$$\boldsymbol{F}(\boldsymbol{w})=\begin{bmatrix}\left(1+\frac{h}{2}\right)y_a-2w_1+\left(1-\frac{h}{2}\right)w_2-h^2\cos w_1\\ \vdots\\ \left(1+\frac{h}{2}\right)w_{i-1}-2w_i+\left(1-\frac{h}{2}\right)w_{i+1}-h^2\cos w_i\\ \vdots\\ \left(1+\frac{h}{2}\right)w_{n-1}-2w_n+\left(1-\frac{h}{2}\right)y_b-h^2\cos w_n\end{bmatrix}.$$

F 的 Jacobi 矩阵 $\mathrm{D}\boldsymbol{F}(\boldsymbol{w})$ 是

$$\begin{bmatrix}-2+h^2\sin w_1 & 1-\frac{h}{2} & 0 & \cdots & 0\\ 1+\frac{h}{2} & -2+h^2\sin w_2 & \ddots & \ddots & \vdots\\ 0 & 1+\frac{h}{2} & \ddots & 1-\frac{h}{2} & 0\\ \vdots & \ddots & \ddots & -2+h^2\sin w_{n-1} & 1-\frac{h}{2}\\ 0 & \cdots & 0 & 1+\frac{h}{2} & -2+h^2\sin w_n\end{bmatrix}.$$

只要对边界条件信息加以适当的改变, 以下代码就能引入程序 7.1 来处理非线性边值问题.

```
function y=f(w)
global h n ya yb
  y=zeros(n,1);
  y(1)=-2*w(1)+(1+h/2)*ya+(1-h/2)*w(2)-h*h*cos(w(1));
  y(n)=(1+h/2)*w(n-1)-2*w(n)-h*h*cos(w(n))+(1-h/2)*yb;
  for j=2:n-1
```

```
    y(j)=-2*w(j)+(1+h/2)*w(j-1)+(1-h/2)*w(j+1)-h*h*cos(w(j));
  end

function a=jac(w)
global h n ya yb
  a=zeros(n,n);
  for j=1:n
    a(j,j)=-2+h*h*sin(w(j));
  end
  for j=1:n-1
    a(j,j+1)=1-h/2;
    a(j+1,j)=1+h/2;
  end
```

图 7-8b 展示了所得到的解曲线 $y(t)$. ◀

计算机问题 7.2

1. 用有限差分法, 求下列线性边值问题的近似解, 取 $n = 9, 19, 39$.

(a) $\begin{cases} y'' = y + \frac{2}{3}e^t, \\ y(0) = 0, \\ y(1) = \frac{1}{3}e; \end{cases}$ **(b)** $\begin{cases} y'' = (2 + 4t^2)y, \\ y(0) = 1, \\ y(1) = e. \end{cases}$

绘出近似解以及准确解 (a) $y(t) = te^t/3$ 和 (b) $y(t) = e^{t^2}$ 的图形, 并在分离的半对数坐标图中把误差展示为 t 的函数.

2. 用有限差分法, 求下列线性边值问题的近似解, 取 $n = 9, 19, 39$.

(a) $\begin{cases} 9y'' + \pi^2 y = 0, \\ y(0) = -1, \\ y\left(\frac{3}{2}\right) = 3; \end{cases}$ **(b)** $\begin{cases} y'' = 3y - 2y', \\ y(0) = e^3, \\ y(1) = 1. \end{cases}$

绘出近似解和准确解 (a) $y(t) = 3\sin\frac{\pi t}{3} - \cos\frac{\pi t}{3}$, (b) $y(t) = e^{3-3t}$ 的图形, 并在分离的半对数坐标图中把误差展示为 t 的函数.

3. 用有限差分法, 求下列非线性边值问题的近似解, 取 $n = 9,19,39$:

(a) $\begin{cases} y'' = 18y^2, \\ y(1) = \frac{1}{3}, \\ y(2) = \frac{1}{12}; \end{cases}$ **(b)** $\begin{cases} y'' = 2e^{-2y}(1 - t^2), \\ y(0) = 0, \\ y(1) = \ln 2. \end{cases}$

绘出近似解和准确解 (a) $y(t) = \frac{1}{3t^2}$; (b) $y(t) = \ln(t^2 + 1)$ 的图形, 并在分离的半对数坐标图中把误差展示为 t 的函数.

4. 用有限差分法, 画出以非线性边值问题的解, 取 $n = 9, 19, 39$:

(a) $\begin{cases} y'' = e^y, \\ y(0) = 1, \\ y(1) = 3; \end{cases}$ **(b)** $\begin{cases} y'' = \sin y', \\ y(0) = 1, \\ y(1) = -1. \end{cases}$

5. **(a)** 用解析方法, 求边值问题 $y'' = y, y(0) = 0, y(1) = 1$ 的解. **(b)** 写出方程的有限差分形式, 并且取 $n = 15$, 画出近似解. **(c)** 通过构造在 $t = \frac{1}{2}$ 处的误差与 $n(n = 2^p - 1, p = 2, \cdots, 7)$ 的双对数坐标图来比较近似解与准确解.

6. 用有限差分法, 解非线性边值问题 $4y'' = ty^4, y(1) = 2, y(2) = 1$. 取 $n = 15$, 画出近似解. 构造在 $t = \frac{3}{2}$ 处的误差与 $n = 2^p(p = 2, \cdots, 7)$ 的双对数坐标图来比较近似解与准确解.
7. 对计算机问题 5 中的近似解进行外推. 把 Richardson 外推技巧 (见 5.1 节) 用到公式 $N(h) = w_h\left(\frac{1}{2}\right)$, 带有步长 h 的有限差分近似. 通过仅仅使用由 $h = \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ 得到近似值外推能与准确解 $y(\frac{1}{2})$ 如何接近?
8. 对计算机问题 6 中的近似解进行外推, 用公式 $N(h) = w_h\left(\frac{3}{2}\right)$, 带有步长 h 的有限差分近似. 通过仅仅使用由 $h = \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ 得到的近似解外推能与准确解 $y\left(\frac{3}{2}\right)$ 如何接近?
9. 用有限差分方法, 解非线性边值问题 $y'' = \sin y, y(0) = 1, y(\pi) = 0$. 对 $n = 9, 19, 39$ 画出近似解.
10. 用有限差分法解方程

$$\begin{cases} y'' = 10y(1-y), \\ y(0) = 0, \\ y(1) = 1, \end{cases}$$

对 $n = 9, 19, 39$ 画出近似解.

11. 对 $c > 0$, 解

$$\begin{cases} y'' = cy(1-y), \\ y(0) = 0, \\ y\left(\dfrac{1}{2}\right) = \dfrac{1}{4}, \\ y(1) = 1, \end{cases}$$

要求精确到 3 位小数.(提示: 先考虑通过固定 3 个边界条件中的两个而形成的边值问题. 设 $G(c)$ 是第 3 个边界条件的偏差, 并用对分法解出 $G(c) = 0$.)

7.3 配置法与有限元法

和有限差分方法一样, 配置法和有限元法的思想是把边值问题化为一组可解的方程. 然而, 不是通过用有限差分代替导数来离散化微分方程, 而是解由某种函数形式给出, 它的参数由方法进行调整.

选择一组基函数 $\varphi_1(t), \cdots, \varphi_n(t)$, 它们可以是多项式、三角函数、样条函数或者其他的简单函数. 然后考虑可能的解

$$y(t) = c_1\varphi_1(t) + \cdots + c_n\varphi_n(t). \tag{7.15}$$

求近似解就转换成确定 c_i 的值. 我们将考虑求系数的两种不同的方法.

配置法是将 (7.15) 代入边值问题并计算在网格点上的值. 这个方法是直接把问题转化成求解一个方程组, 如果原问题是线性的, 那么这个方程就是线性的. 每一个点给出一个方程, 并且从其中解出 c_i, 这是一类插值法.

第二种近似方法即有限元法, 并不进行插值, 而是着手把拟合作为最小二乘问题来处理. 有限元法采用 Galerkin 投影, 在平方误差意义下把 (7.15) 和准确解之间的差极小化. 下面探讨这两种方法.

7.3.1 配置法

选取 n 个点, 包括起点 a 终点 b, 譬如

$$a = t_1 < t_2 < \cdots < t_n = b. \tag{7.16}$$

配置法通过把候选解 (7.15) 代入微分方程并计算在点 (7.16) 的微分方程从而得到含 n 个未知数 $c_1, c_2, \cdots, c_n$ 的 n 个方程.

例 7.11 用配置法解边值问题:

$$\begin{cases} y'' = 4y, \\ y(0) = 1, \\ y(1) = 3. \end{cases}$$

一开始, 我们尽可能简单, 选择基函数 $\varphi_j(t) = t^{j-1} (1 \leqslant j \leqslant n)$. 解的形式将是

$$y(t) = \sum_{j=1}^{n} c_j \varphi_j(t) = \sum_{j=1}^{n} c_j t^{j-1}. \tag{7.17}$$

我们将写出含 n 个未知数 $c_1, c_2, \cdots, c_n$ 的 n 个方程. 第一个方程和最后一个方程就是边界条件

$$i = 1: \quad c_1 = \sum_{j=1}^{n} c_j \varphi_j(0) = y(0) = 1,$$

$$i = n: \quad c_1 + \cdots + c_n = \sum_{j=1}^{n} c_j \varphi_j(1) = y(1) = 3.$$

其余的 $n-2$ 个方程是通过计算微分方程在 $t_i (2 \leqslant i \leqslant n-1)$ 处的值而得到. 把 $y(t) = \sum_{j=1}^{n} c_j t^{j-1}$ 代入微分方程得到

$$\sum_{j=1}^{n} (j-1)(j-2) c_j t^{j-3} - 4 \sum_{j=1}^{n} c_j t^{j-1} = 0.$$

对每一个 i, 在 t_i 处计算上式得到

$$\sum_{j=1}^{n} [(j-1)(j-2) t_i^{j-3} - 4 t_i^{j-1}] c_j = 0.$$

以上 n 个方程组成一个线性方程组 $\boldsymbol{Ac}=\boldsymbol{g}$, 其中系数矩阵 $\boldsymbol{A}$ 的元素定义为

$$A_{ij}=\begin{cases}1\quad 0\quad 0\quad \cdots\quad 0, & \text{第 1 行},\\ (j-1)(j-2)t_i^{j-3}-4t_i^{j-1}, & \text{第 2 行到第} n-1 \text{行},\\ 1\quad 1\quad 1\quad \cdots\quad 1, & \text{第} n \text{行},\end{cases}$$

而 $g=(1,0,0,\cdots,0,3)^{\mathrm{T}}$. 通常用等距网格点

$$t_i=a+\frac{i-1}{n-1}(b-a)=\frac{i-1}{n-1}.$$

在解出 c_j 后, 就得到了近似解 $y(t)=\sum c_j t^{j-1}$.

对 $n=2$, 方程组 $\boldsymbol{Ac}=\boldsymbol{g}$ 是

$$\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}c_1\\ c_2\end{bmatrix}=\begin{bmatrix}1\\ 3\end{bmatrix},$$

并且它的解是 $c=(1,2)^{\mathrm{T}}$. (7.17) 的近似解是直线 $y(t)=c_1+c_2t=1+2t$. 对 $n=4$ 进行计算得到近似解 $y(t)\approx 1-0.188\,6t+1.027\,3t^2+1.161\,3t^3$. 在图 7-9 中画出了 $n=2$ 和 $n=4$ 的解. 图 7-3b 显示当 $n=4$ 时近似解已经非常接近准确解 (7.4). 随着 n 的增大而更加精确. ◀

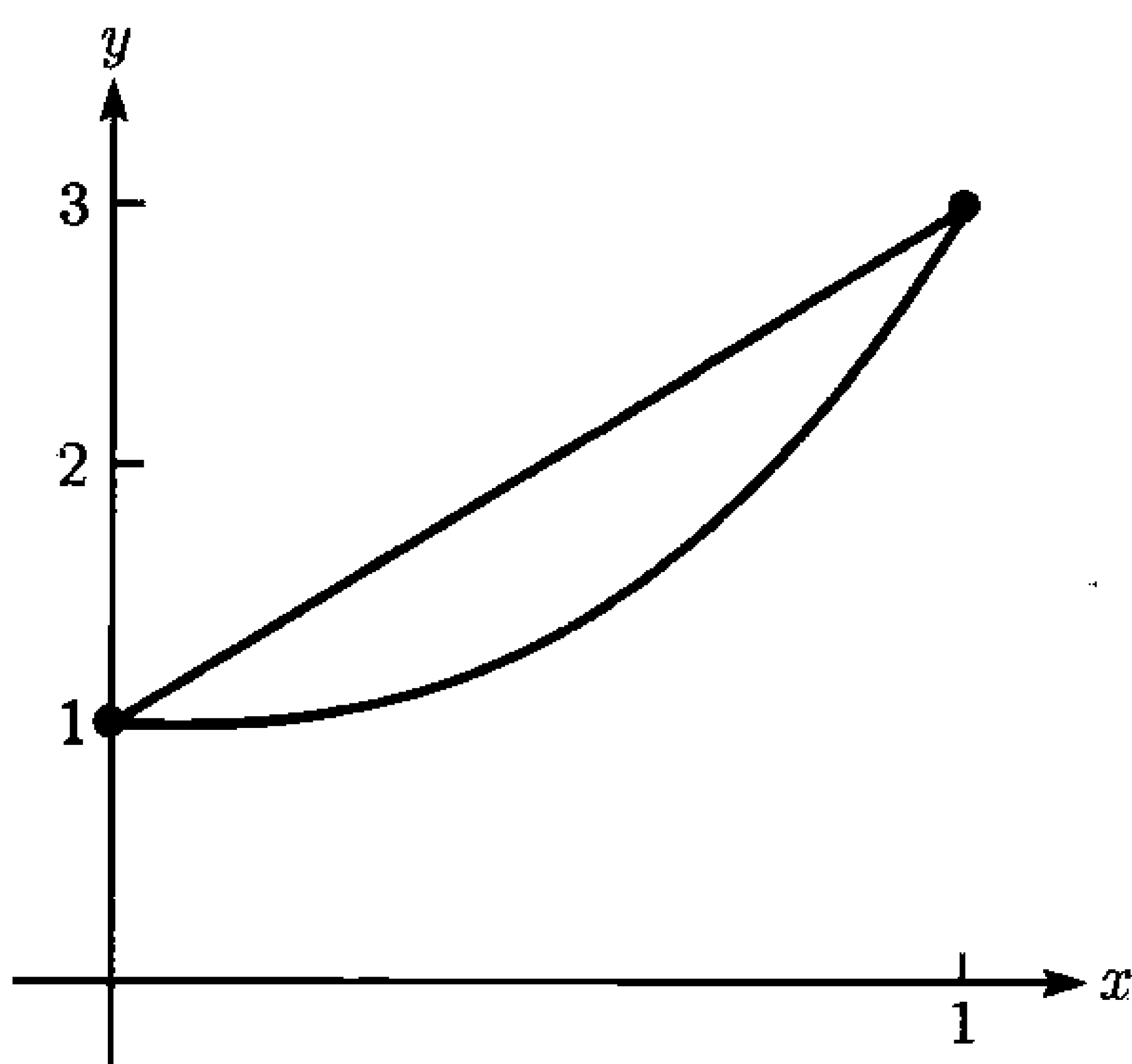


图 7-9 用配置法解例 7.11 的线性边值问题: 给出 $n=2$ 和 $n=4$ 时的解

在例 7.11 中, 因为微分方程是线性的, 所求解的关于 c_i 的方程组也是线性的. 类似地, 能用配置法解非线性边值问题. Newton 法用于求解所得到的非线性方程组, 正如在有限差分方法中一样.

尽管为了简单, 我们叙述了取单项式为基函数的配置法的使用, 但是还有许多更好的选择. 一般不推荐多项式基函数. 因为配置法本质上是对解进行插值, 而多项式基函数的使用会造成方法对 Runge 现象 (见第 3 章) 的敏感. 事实上,n 很大

时, 若函数使线性方程组的系数矩阵成为病态的, 则单项式基函数 t^j 彼此不正交. 不用等距点而用 Chebyshev(契比雪夫) 多项式的根作为求值点, 能改进系数矩阵的条件作用.

选择三角函数为基函数导致 Fourier 分析或**谱方法**(spectral method), 它们大量地用于边值问题和偏微分方程中. 这是一种 "全局" 方法, 其中基函数在 t 的大范围内非零, 但是有好的正交性. 我们将在第 10 章中研究离散 Fourier 逼近.

选择样条作为基函数导出**有限元法**(finite element method). 在这种方法中, 每一个基函数仅在 t 的小范围内非零. 有限元法大量地用于较高维的边值问题和偏微分方程中, 特别在非规则边界使得用标准基函数进行参数化很不方便时, 更是如此.

7.3.2 有限元和 Galerkin 方法

在配置法中, 假设函数的形式为 $y(t) = \sum c_i \varphi_i(t)$, 而且通过使解满足边界条件以及在离散点上精确满足微分方程而求得解. 另一方面, Galerkin 方法把微分方程沿着解的误差的平方极小化. 这就导出了关于 c_i 的一个不同的方程组.

考虑边值问题

$$\begin{cases} y'' = f(t, y, y'), \\ y(a) = y_a, \\ y(b) = y_b, \end{cases}$$

我们的目标是选取近似解 y, 使得**残差** $r = y'' - f$, 即微分方程两边的差, 尽可能地小. 与第 4 章中的最小二乘法类似, 这是通过选取 y 使得残差与潜在解的向量空间正交而实现的.

对区间 $[a, b]$, 定义平方可积函数的向量空间:

$$L^2[a,b] = \left\{ \text{函数} y(t) \text{在} [a,b] \text{上} \,\middle|\, \int_a^b y(t)^2 \mathrm{d}t \text{存在且有限} \right\}.$$

L^2 函数空间有**内积**

$$\langle y_1, y_2 \rangle = \int_a^b y_1(t) y_2(t) \mathrm{d}t,$$

它具有通常的性质:

(1) $\langle y_1, y_1 \rangle > 0$, 当且仅当 $y_1 = 0$ 时等式成立;

(2) $\langle \alpha y_1 + \beta y_2, z \rangle = \alpha \langle y_1, z \rangle + \beta \langle y_2, z \rangle$, 其中 α, β 是标量;

(3) $\langle y_1, y_2 \rangle = \langle y_2, y_1 \rangle$.

如果 $\langle y_1, y_2 \rangle = 0$, 那么函数 y_1 和 y_2 在 $L^2[a, b]$ 内**正交**. 由于 $L^2[a, b]$ 是无穷维向量空间, 不可能通过有限次计算使残差 $r = y'' - f$ 与 $L^2[a, b]$ 中的所有向量正

交. 然而, 可以选取一组基, 它们用现有的计算资源张成 L^2 的尽可能大的子空间. 设 $n+2$ 个基函数集合表示为 $\varphi_0(t), \cdots, \varphi_{n+1}(t)$, 以后我们将确定这些基函数.

Galerkin 方法包含两个主要想法. 第一个想法就是通过在 L^2 内积意义下使残差 r 与所有的基函数正交, 从而使 r 极小化. 这意味着使 $\int_a^b (y''-f)\varphi_i \mathrm{d}t = 0$ 或者

$$\int_a^b y''(t)\varphi_i(t)\mathrm{d}t = \int_a^b f(t,y,y')\varphi_i(t)\mathrm{d}t, 0 \leqslant i \leqslant n+1. \tag{7.18}$$

我们称形式 (7.18) 为边值问题的**弱形式**(weak form).

Galerkin 方法的第二个想法是用分部积分消去二阶导数. 注意到

$$\begin{aligned}\int_a^b y''(t)\varphi_i(t)\mathrm{d}t &= \varphi_i(t)y'(t)\Big|_a^b - \int_a^b y'(t)\varphi_i'(t)\mathrm{d}t \\ &= \varphi_i(b)y'(b) - \varphi_i(a)y'(a) - \int_a^b y'(t)\varphi_i'(t)\mathrm{d}t,\end{aligned} \tag{7.19}$$

并使用 (7.18) 和 (7.19) 便给出一组方程

$$\int_a^b f(t,y,y')\varphi_i(t)\mathrm{d}t = \varphi_i(b)y'(b) - \varphi_i(a)y'(a) - \int_a^b y'(t)\varphi_i'(t)\mathrm{d}t. \tag{7.20}$$

在函数形式

$$y(t) = \sum_{i=0}^{n+1} c_i\varphi_i(t) \tag{7.21}$$

下, 对每一个 i, 能解出 c_i.

Galerkin 的这两个想法使得用极其简单的函数作为有限元素 $\varphi_i(t)$ 很方便. 我们将仅仅介绍分段线性 B 样条, 并指导读者去参阅有关更高等的著作.

从 t 轴上的网格点 $t_0 < t_1 < \cdots < t_n < t_{n+1}$ 开始. 对 $i = 1, \cdots, n$, 定义

$$\varphi_i(t) = \begin{cases} \dfrac{t-t_{i-1}}{t_i - t_{i-1}}, & t_{i-1} < t \leqslant t_i, \\ \dfrac{t_{i+1}-t}{t_{i+1}-t_i}, & t_i < t < t_{i+1}, \\ 0, & \text{其他}, \end{cases}$$

并定义

$$\varphi_0(t) = \begin{cases} \dfrac{t_1 - t}{t_1 - t_0}, & t_0 \leqslant t < t_1, \\ 0, & \text{其他}, \end{cases} \qquad \varphi_{n+1}(t) = \begin{cases} \dfrac{t - t_n}{t_{n+1} - t_n}, & t_n < t \leqslant t_{n+1}, \\ 0, & \text{其他}, \end{cases}$$

图 7-10 中表示的分段线性 “帐篷” 函数 φ_i 满足以下有趣的性质:

$$\varphi_i(t_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{7.22}$$

对已知点集 $(t_i, c_i), i = 0, 1, \cdots, n+1$, 定义**分段线性 B 样条**

$$S(t) = \sum_{i=0}^{n+1} c_i \varphi_i(t).$$

根据 (7.22) 立刻得到 $S(t_j) = \sum_{i=0}^{n+1} c_i \varphi_i(t_j) = c_j$. 因此, $S(t)$ 是在已知点 (t_i, c_i) 插值的分段线性函数. 换言之, y 坐标就是系数! 这使解 (7.21) 的插值简单化. c_i 不仅是系数, 而且也是解在网格点 t_i 处的值.

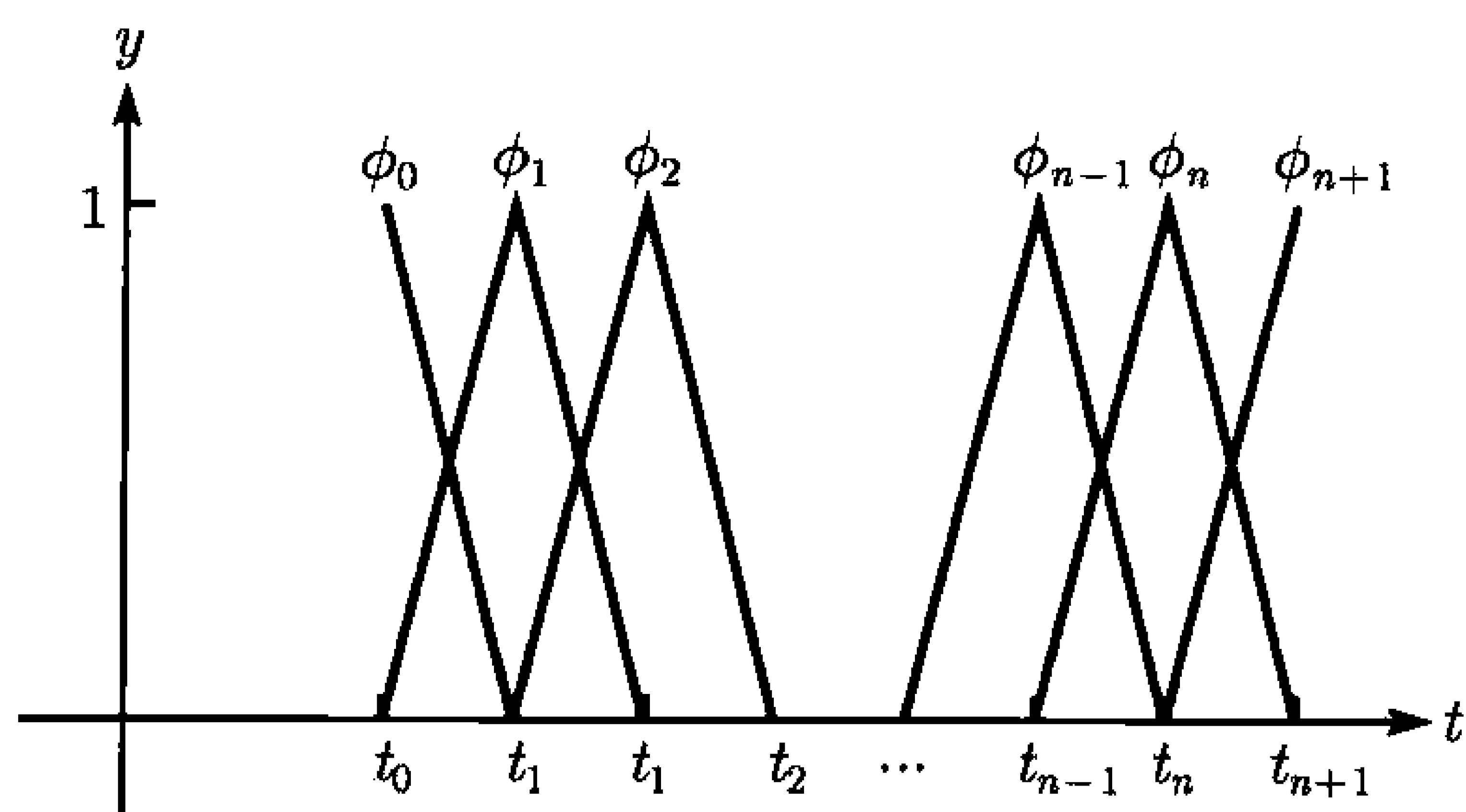


图 7-10 把分段线性 B 样条用作有限元素: 对 $1 \leqslant i \leqslant n$, 每一个 $\varphi_i(t)$ 在区间 $[t_{i-1}, t_{i+1}]$ 上都有支撑

例 7.12 用有限元法解边值问题

$$\begin{cases} y'' = 4y, \\ y(0) = 1, \\ y(1) = 3. \end{cases}$$

亮点 正交性

我们在第 4 章看到, 一点到一个平面的距离通过画一条从该点到这个平面的垂直线段而极小化. 该平面代表逼近这个点的候选者, 其间的距离就是逼近误差. 这个关于正交性的简单事实遍及数值分析. 它是最小二乘逼近的核心, 并且也是边值问题和偏微分方程的 Galerkin 方法以及 Gauss 求积 (第 5 章)、压缩 (见第 10 章和第 11 章) 和特征值问题的解 (第 12 章) 的基础.

设 $\varphi_0, \cdots, \varphi_{n+1}$ 是在 $[a, b]$ 上的一组网格点上的分段线性 B 样条, 如图 7-10 所示. 它们将作为 Galerkin 方法的基函数.

设 c_i 中的第一个和最后一个从配置法已得到

$$1 = y(0) = \sum_{i=0}^{n+1} c_i \varphi_i(0) = c_0 \varphi_0(0) = c_0,$$

$$3 = y(1) = \sum_{i=0}^{n+1} c_i\varphi_i(1) = c_{n+1}\varphi_{n+1}(1) = c_{n+1}.$$

对 $i = 1, \cdots, n$, 用有限元方程 (7.20):

$$\int_0^1 f(t, y, y')\varphi_i(t)\mathrm{d}t + \int_0^1 y'(t)\varphi_i'(t)\mathrm{d}t = 0$$

注意 (7.20) 的边界项对 $i = 1, \cdots, n$ 是零.

现在把函数形式 $y(t) = \sum c_i\varphi_i(t)$ 代入上式并利用微分方程 $f(t, y, y') = 4y$ 就得到

$$\begin{aligned}0 &= \int_0^1 \left(4\varphi_i(t)\sum_{j=0}^{n+1} c_j\varphi_j(t) + \sum_{j=0}^{n+1} c_j\varphi_j'(t)\varphi_i'(t)\right)\mathrm{d}t\\ &= \sum_{j=0}^{n+1} c_j\left[4\int_0^1 \varphi_i(t)\varphi_j(t)\mathrm{d}t + \int_0^1 \varphi_j'(t)\varphi_i'(t)\mathrm{d}t\right].\end{aligned}$$

假设网格是步长为 h 的等步长网格. 我们将需要以下积分 $(i = 1, \cdots, n)$:

$$\begin{aligned}\int_a^b \varphi_i(t)\varphi_{i+1}(t)\mathrm{d}t &= \int_0^h \frac{t}{h}\left(1 - \frac{t}{h}\right)\mathrm{d}t\\ &= \int_0^h \left(\frac{t}{h} - \frac{t^2}{h^2}\right)\mathrm{d}t = \frac{t^2}{2h} - \frac{t^3}{3h^2}\Big|_0^h = \frac{h}{6},\end{aligned} \tag{7.23}$$

$$\int_a^b (\varphi_i(t))^2\mathrm{d}t = 2\int_0^h \left(\frac{t}{h}\right)^2\mathrm{d}t = \frac{2}{3}h, \tag{7.24}$$

$$\int_a^b \varphi_i'(t)\varphi_{i+1}'(t)\mathrm{d}t = \int_0^h \frac{1}{h}\left(-\frac{1}{h}\right)\mathrm{d}t = -\frac{1}{h}, \tag{7.25}$$

$$\int_a^b (\varphi_i'(t))^2\mathrm{d}t = 2\int_0^h \left(\frac{1}{h}\right)^2\mathrm{d}t = \frac{2}{h}. \tag{7.26}$$

对 B 样条, 使用关系式 (7.23~7.26) $(i = 1, 2, \cdots, n)$, 发现方程是

$$\begin{aligned}\left[\frac{2}{3}h - \frac{1}{h}\right]c_0 + \left[\frac{8}{3}h + \frac{2}{h}\right]c_1 + \left[\frac{2}{3}h - \frac{1}{h}\right]c_2 &= 0,\\ \left[\frac{2}{3}h - \frac{1}{h}\right]c_1 + \left[\frac{8}{3}h + \frac{2}{h}\right]c_2 + \left[\frac{2}{3}h - \frac{1}{h}\right]c_3 &= 0,\\ &\vdots\\ \left[\frac{2}{3}h - \frac{1}{h}\right]c_{n-1} + \left[\frac{8}{3}h + \frac{2}{h}\right]c_n + \left[\frac{2}{3}h - \frac{1}{h}\right]c_{n+1} &= 0.\end{aligned} \tag{7.27}$$

注意到, $c_0 = y_a = 1$ 及 $c_{n+1} = y_b = 3$, 所以方程组的矩阵形式是对称三对角的:

$$\begin{bmatrix} \alpha & \beta & 0 & \cdots & 0 \\ \beta & \alpha & \ddots & \ddots & \vdots \\ 0 & \beta & \ddots & \beta & 0 \\ \vdots & \ddots & \ddots & \alpha & \beta \\ 0 & \cdots & 0 & \beta & \alpha \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} -y_a\beta \\ 0 \\ \vdots \\ 0 \\ -y_b\beta \end{bmatrix},$$

这里 $\alpha = \frac{8}{3}h + \frac{2}{h}, \beta = \frac{2}{3}h - \frac{1}{h}$.

调用第 2 章用过的 MATLAB命令spdiags, 可以写出一个非常简洁的稀疏的执行过程.

```
% Program 7.2 Finite element solution of linear BVP.
% Inputs: interval inter, boundary values bv, number of steps n
% Output: solution values c
Example usage: c=bvpfem ([0 1],[1 3],9);
function c=bvpfem(inter,bv,n)
a=inter(1); b=inter(2); ya=bv(1); yb=bv(2);
h=(b-a)/(n+1);
alpha=(8/3)*h+2/h; beta=(2/3)*h-1/h;
e=ones(n,1);
M=spdiags([beta*e alpha*e beta*e],-1:1,n,n);
d=zeros(n,1);
d(1)=-ya*beta;
d(n)=-yb*beta;
c=M\d;
```

对 $n = 3$, MATLAB代码给出表 7-2 中的 c_i.

表 7-2

i	t_i	$w_i = c_i$	y_i
0	0.00	1.000 0	1.000 0
1	0.25	1.010 9	1.018 1
2	0.50	1.285 5	1.296 1
3	0.75	1.895 5	1.904 9
4	1.00	3.000 0	3.000 0

注意近似解 w_i 在 t_i 处有值 c_i, 它可与准确解 y_i 比较.

其误差大概是 10^{-2}, 与有限差分法的误差相同. 事实上, 图 7-11 表明, 对大的 n 值运行有限元法给出的收敛曲线与图 7-7 中的有限差分法的几乎一致, 都表明了 $O(n^{-2})$ 收敛性. ◀

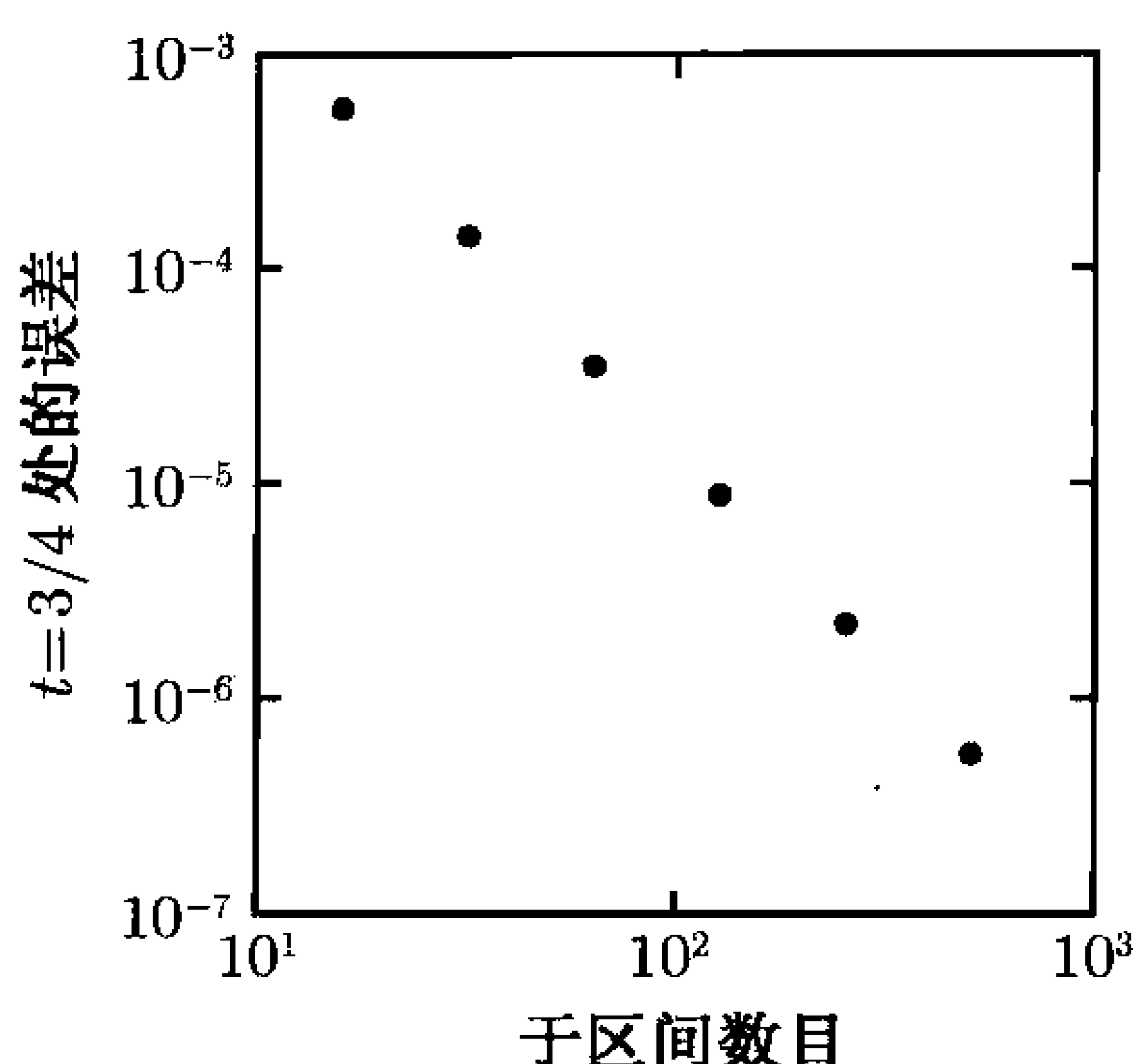


图 7-11 有限元方法的收敛性. 对例 7.12, 在 $t_i = 3/4$ 处的误差 $|w_i - y_i|$ 当作子区间数目 n 的函数. 按照斜率, 误差是 $O(n^{-2}) = O(h^2)$

计算机问题 7.3

1. 取 $n = 8$ 和 $n = 16$, 用配置法求以下线性边值问题的近似解.

$$\text{(a)}\begin{cases} y'' = y + \frac{2}{3}\mathrm{e}^t, \\ y(0) = 0, \\ y(1) = \frac{1}{3}\mathrm{e}; \end{cases} \qquad \text{(b)}\begin{cases} y'' = (2 + 4t^2)y, \\ y(0) = 1, \\ y(1) = \mathrm{e}. \end{cases}$$

画出近似解及正确解 (a) $y(t) = \frac{t\mathrm{e}^t}{3}$ 及 (b) $y(t) = \mathrm{e}^{t^2}$, 并在分离的半对数坐标图中把误差展现为 t 的函数.

2. 取 $n = 8$ 及 n =16, 用配置法求以下线性边值问题的近似解.

$$\text{(a)}\begin{cases} 9y'' + \pi^2 y = 0, \\ y(0) = -1, \\ y\left(\frac{3}{2}\right) = 3; \end{cases} \qquad \text{(b)}\begin{cases} y'' = 3y - 2y', \\ y(0) = \mathrm{e}^3, \\ y(1) = 1. \end{cases}$$

画出近似解及准确解 (a) $y(t) = 2\sin\frac{\pi t}{3} - \frac{\cos\pi t}{3}$ 及 (b) $y(t) = \mathrm{e}^{3-3t}$. 并在分离的半对数坐标图中把误差展现为 t 的常数.

3. 用有限元法执行计算机问题 1 的步骤.

4. 用有限元法执行计算机问题 2 的步骤.

软件和进一步阅读

在许多关于常微分方程的教科书中讨论了边值问题. 教科书 [1] 是对常微分方程边值问题的技巧的综述, 它包括本章没有讲述的多重打靶法. 关于边值问题的打靶法和有限差分法的其他好的参考文献包括 [5,6,7] .

IMSL 库的程序 BVPMS 和 BVPFD 对两点边值问题分别执行打靶法和有限差分法. BVPFD 使用一种变阶、变步长有限差分方法.

NAG 程序 D02HAF 对两点边值问题用 Runge-Kutta-Merson 方法和 Newton 迭代执行打靶法. 程序 D02GAF 用 Newton 迭代执行一种有限差分技术去解结果方程. Jacobi 矩阵是通过数值微分计算的. D02JAF 通过配置法解一个 n 阶常微分方程的线性边值问题.

Netlib 库包含用户可调用的两种 Fortran 子程序: 解决线性问题的 MUSL 和解决非线性问题的 MUSN. 每一种都是以打靶法为基础的.

第 8 章　偏微分方程

20 世纪 70 年代由英特尔公司生产的 8086 中央处理器以 5MHz 的速度运行, 需要的能量小于 5W. 如今, 随着芯片速度几百倍地增加, 其耗能超过了 50W. 为避免过高温度对处理器的损伤, 使用散热器和风扇来散发热量是必不可少的. 在遵循 Moore 定律以得到更快的处理速度时, 如何降温始终是个障碍.

抛物型偏微分方程是热量散发时间过程很好的模型. 当热量达到一个平衡时, 一个椭圆型方程模拟了定常 (稳态) 分布.

实例检验　8.3.1 节后的实例检验 8, 利用带有热对流边界条件的椭圆型偏微分方程, 给出了如何模拟一个简单的散热器配置.

偏微分方程是带有一个以上的自变量的微分方程. 然而这个主题太广泛, 我们将把讨论局限于带有两个自变量的方程, 其形式为

$$Au_{xx} + Bu_{xy} + Cu_{yy} + F(u_x, u_y, u, x, y) = 0, \tag{8.1}$$

这里对自变量的偏导数由下标 x 和 y 表示, u 表示解. 当一个自变量表示时间, 如在热方程中, 可以把自变量称为 x 和 t.

(8.1) 中的前面 3 项决定解的各种性质. 含有两个自变量的二阶偏微分方程分类如下.

(1) 抛物型, 如果 $B^2 - 4AC = 0$;

(2) 双曲型, 如果 $B^2 - 4AC > 0$;

(3) 椭圆型, 如果 $B^2 - 4AC < 0$.

上面 3 类方程的实际区别是, 抛物型和双曲型方程定义在开域. 关于一个自变量 (大多数情形是时间变量) 的边界条件给定在区域的一端, 而离开边界求得方程的解. 另一方面, 椭圆型方程习惯上在一个闭域的整个边界上给定边界条件. 我们将研究每种类型的一些例子, 并说明求近似解的数值方法.

8.1　抛物型偏微分方程

热方程

$$u_t = cu_{xx} \tag{8.2}$$

表示沿着一维均匀杆测量得到的温度 u. 常数 $c>0$ 叫做**扩散系数** (diffusion coefficient), 表示制造杆的材料的热扩散率. 热方程是热量从密度较高的区域向密度较低的区域传播的模型. 自变量是 x 和 t.

在 (8.2) 中我们用 t 替代 y, 因为它代表时间. 根据前面的分类, 有 $B^2-4AC=0$, 所以热方程是抛物型的. 所谓的热方程只是作为某种物质扩散的模型的**扩散方程** (diffusion equation) 的一个例子. 在材料科学里, 把同样的方程当作是 Fick 第二定律, 它描述了某一介质中的物质的扩散.

类似于常微分方程的情形, 偏微分方程 (8.2) 有无穷多个解, 因此需要额外的条件来定出特解. 第 6 章和第 7 章处理了常微分方程求解, 那里分别用到初始条件和边界条件. 为了适当地提出一个偏微分方程, 可能会用到初始条件和边界条件的各种结合.

对于热方程, 根据常识, 我们直觉上知道它需要什么条件. 为了唯一地确定该情形, 需要知道沿着杆的初始温度的分布, 以及当时间进展时, 在杆的两端发生了什么. 在有限区间上适当提出的热方程形式是

$$\begin{cases} u_t = cu_{xx}, & a \leqslant x \leqslant b \quad t \geqslant 0, \\ u(x,0) = f(x), & a \leqslant x \leqslant b, \\ u(a,t) = l(t), & t \geqslant 0, \\ u(b,t) = r(t), & t \geqslant 0, \end{cases} \tag{8.3}$$

这里杆沿着区间 $a \leqslant x \leqslant b$ 延伸. 扩散系数 c 控制热传导速度. 在 $[a,b]$ 上的函数 $f(x)$ 给出沿着杆的初始温度分布, $l(t)$ 和 $r(t)$ 对 $t \geqslant 0$ 给出两端的温度. 这里我们已经把初始条件 $f(x)$ 和边界条件 $l(t)$ 及 $r(t)$ 结合起来, 从而确定偏微分方程的唯一解.

8.1.1 前向差分方法

根据前两章建立的方法, 用有限差分方法来求偏微分方程的近似解. 想法是对自变量建立网格并把偏微分方程离散化. 把连续问题变成有限多个方程的离散问题. 如果这个偏微分方程是线性的, 那么它的离散方程是线性的, 因此, 能够用第 2 章的方法求解.

为在时间区间 $[0,T]$ 上离散热方程, 我们考虑图 8-1 所示的点的网格. 其中, 实心圆表示由初始条件和边界条件而已知的解 $u(x,t)$ 的值, 而空心圆是网格点, 并将通过这种方法填成实心. 我们用 $u(x_i,t_j)$ 表示在 (x_i,t_j) 处的准确解, 因而 w_{ij} 表示它的近似解. 令 M 和 N 表示 x 方向和 t 方向的总步数, 令 $h=(b-a)/M$, $k=T/N$ 是 x 方向和 t 方向的网格步长.

第 5 章中的离散公式可用来对 x 方向和 t 方向的导数进行近似. 例如, 用关于

x 的二阶导数的中心差分公式就得到

$$u_{xx}(x,t) \approx \frac{1}{h^2}\left(u(x+h,t) - 2u(x,t) + u(x-h,t)\right), \tag{8.4}$$

它的误差是 $h^2 u_{xxxx}(c_1,t)/12$; 而关于时间变量, 对一阶导数用前向差商公式给出

$$u_t(x,t) \approx \frac{1}{k}\left(u(x,t+k) - u(x,t)\right), \tag{8.5}$$

它的误差是 $ku_{tt}(x,c_2)/2$, 其中 $x-h < c_1 < x+h, t < c_2 < t+h$. 把它们代入在点 (x_i,t_j) 处的热方程就得到

$$\frac{c}{h^2}(w_{i+1,j} - 2w_{ij} + w_{i-1,j}) \approx \frac{1}{k}(w_{i,j+1} - w_{ij}), \tag{8.6}$$

它的局部截断误差由 $O(k)+O(h^2)$ 给出. 恰如在常微分方程中的研究一样, 只要方法稳定, 局部截断误差就能对总体误差给出了一个好的图景. 在提出实施细节后, 我们将研究有限差分方法的稳定性.

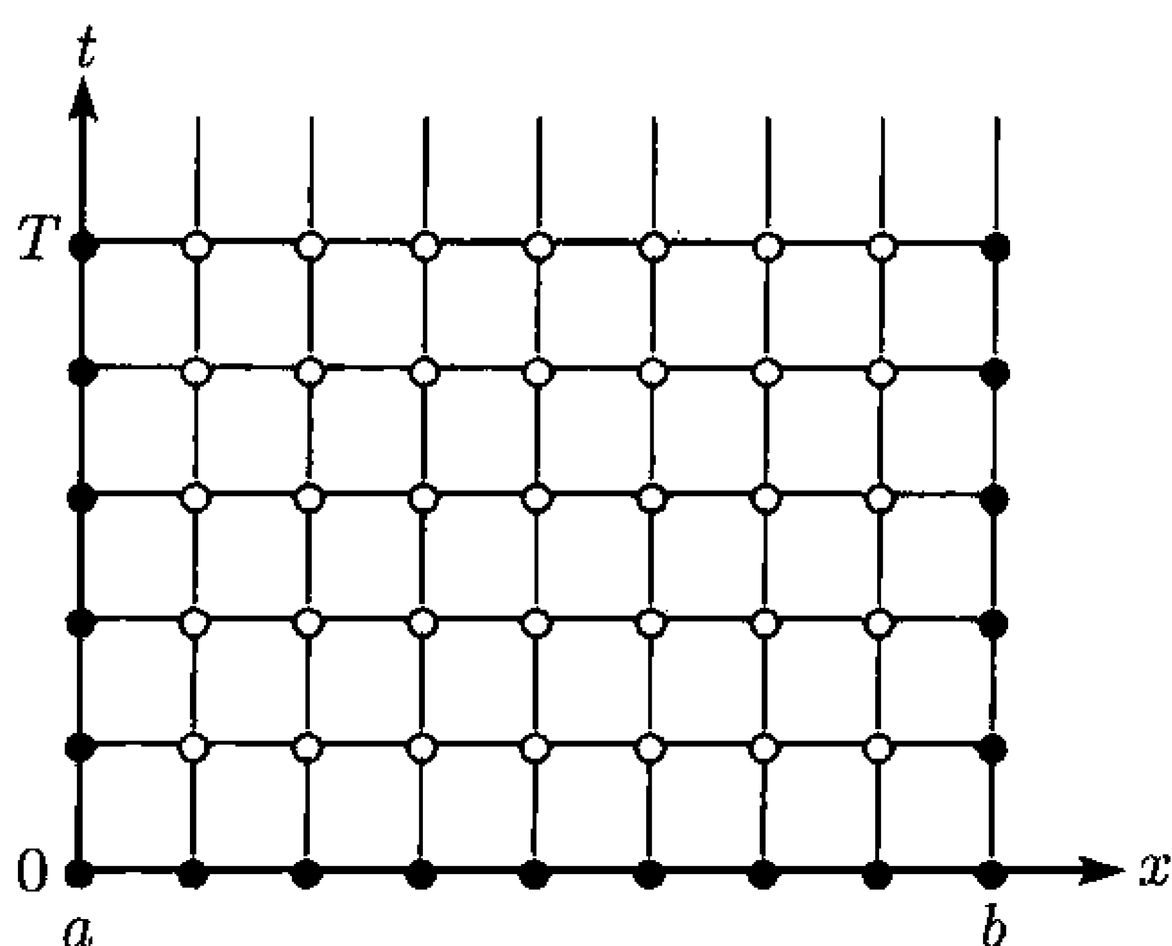


图 8-1 有限差分方法的网格: 实心圆表示已知的初始条件和边界条件, 空心圆表示能确定的未知值

注意初始条件和边界条件给出已知量 $w_{i0}(i=0,\cdots,M), w_{0j}$ 和 $w_{Mj}, j=0,\cdots,N$, 它们相应于图 8-1 中矩形的底边和两腰. 离散形式 (8.6) 可以按时间方向向前逐步求解. 把 (8.6) 重新写成

$$w_{i,j+1} = w_{ij} + \frac{ck}{h^2}\left(w_{i+1,j} - 2w_{ij} + w_{i-1,j}\right) = \sigma w_{i+1,j} + (1-2\sigma)w_{ij} + \sigma w_{i-1,j}, \tag{8.7}$$

这里我们定义 $\sigma = \frac{ck}{h^2}$. 图 8-2 表示 (8.7) 所涉及的一组网格点, 常称为方法的空格样板 (stencil).

前向差分方法 (8.7) 是**显式的**, 这是因为这种方法直接由前面 (在时间意义下) 已知的值来确定新的值. 不是显式的方法就称为**隐式的**. 图 8-2 中这种方法的 “空格样板” 表示它是显式的. 使用矩阵, 可以通过矩阵乘法 $\boldsymbol{w}_{j+1} = \boldsymbol{A}\boldsymbol{w}_j + \boldsymbol{s}_j$, 即

$$\begin{bmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{bmatrix} = \begin{bmatrix} 1-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 1-2\sigma & \sigma & \ddots & \vdots \\ 0 & \sigma & 1-2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma \\ 0 & \cdots & 0 & \sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{bmatrix} + \sigma \begin{bmatrix} w_{0,j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix} \tag{8.8}$$

来得到在时间 t_{j+1} 处的值 $w_{i,j+1}$. 这里 $\boldsymbol{A}$ 是 $m \times m$ 矩阵, $m = M - 1$. 右端向量 $\boldsymbol{s}_j$ 表示由问题强加的边界条件, 本例中就是杆端的温度.

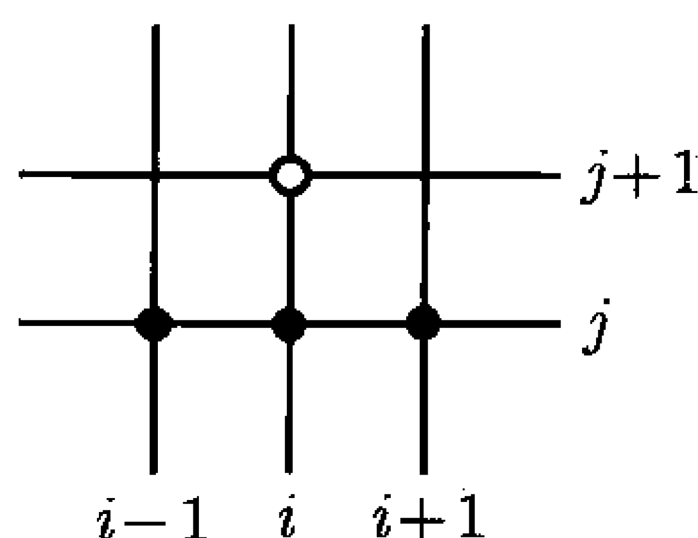
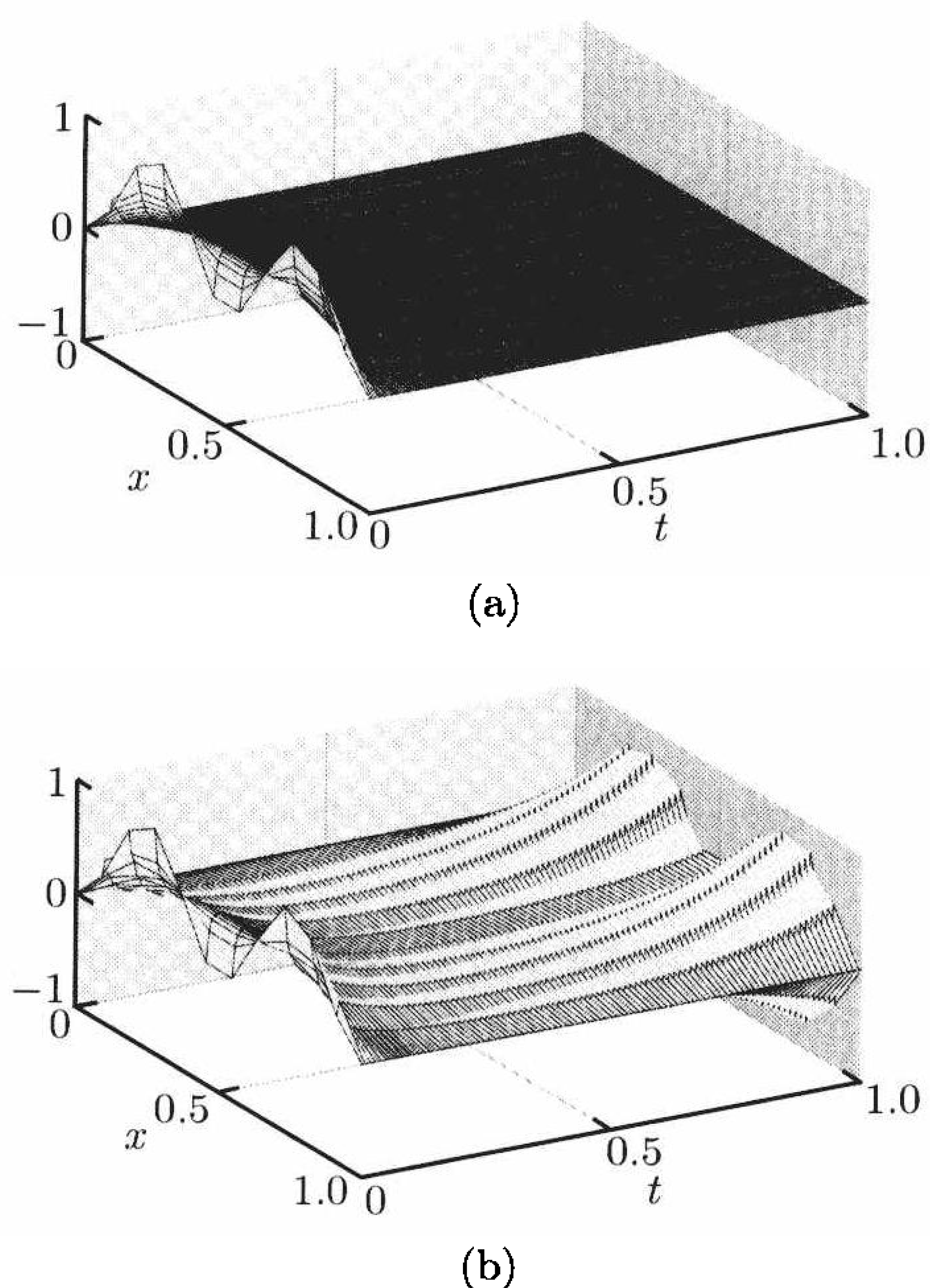


图 8-2 前向差分方法的空格样板: 空心圆表示 $w_{i,j+1}$, 根据 (8.7), 它可以由在实心圆处的值 $w_{i-1,j}, w_{ij}$ 和 $w_{i+1,j}$ 来确定

求解便化成矩阵公式的迭代, 它使我们逐行填满图 8-1 中空心的圆. 迭代矩阵公式 $\boldsymbol{w}_{j+1} = \boldsymbol{A}\boldsymbol{w}_j + \boldsymbol{s}_j$ 类似于第 2 章中讲的线性方程组的迭代方法. 那里我们知道迭代的收敛性依赖于矩阵的特征值. 在目前的情形下, 我们对特征值感兴趣的理由稍有不同, 是为了分析误差的放大或不稳定性.

为了了解误差放大可能是一个问题, 考虑当 $c = 1$ 时, 带有初始条件 $f(x) = \sin^2 2\pi x$ 和边界条件 $u(0,t) = u(1,t) = 0$(对一切 t) 的热方程. 初始温度峰值随着时间扩散, 便产生了图 8-3a 所示的图. 在该图中, 公式 (8.8) 沿着杆使用步长 $h = 0.1$, 而时间步长 $k = 0.004$. 显式前向差分方法 (8.7) 给出图 8-3a 中的近似解, 表明平滑热流在不到一个时间单位后就几乎趋于平衡状态. 这对应了当 $t \to \infty$ 时杆的温度 $u \to 0$.

在图 8-3b 中, 用稍微大一点的时间步长 $k = 0.005\,5$. 首先, 如我们预料的那样, 热扰动一开始就逐渐减少; 但是在较多时间步后, 近似解的微小误差由于用了前向差分而被放大了, 它使解离开正确的平衡状态零而去. 这是求解过程中的人工产物, 也是这种方法不稳定的一种信号. 若让这种模拟继续进行下去, 则这些误差将不断增大. 因此我们被迫要求时间步长相当小以保证收敛. 程序 8.1 给出了执行 (8.8) 中计算的 MATLAB 代码, 步长 $k = 0.004$ 对应于在时间方向 $N = 250$ 步, 从 $t = 0$ 到 $t = 1$.



(a)

(b)

图 8-3 用前向有限差分方法的程序 8.1 求解热方程 (8.2). 参数 $c = 1$, 初始条件 $f(x) = \sin^2 2\pi x$. 空间步长 $h = 0.1$. 当时间步长 $k = 0.004\ 0$ 时, 前向差分方法稳定, 见 (a), 当时间步长 $k = 0.005\ 5$ 时前向差分方法不稳定, 见 (b)

```
% Program 8.1 Forward difference method for heat equation
% input: space interval [xl,xr], time interval [yb,yt],
%        number of space steps M, number of time steps N
% output: solution w
% Example usage: w=heatfd(0,1,0,1,10,250)
function w=heatfd(xl,xr,yb,yt,M,N)
c=1;                                % diffusion coefficient
h=(xr-xl)/M; k=(yt-yb)/N; m=M-1; n=N;
sigma=c*k/(h*h);
a=diag(1-2*sigma*ones(m,1))+diag(sigma*ones(m-1,1),1);
a=a+diag(sigma*ones(m-1,1),-1);     % define matrix a
lside=l(yb+(0:n)*k); rside=r(yb+(0:n)*k);
w(:,1)=f(xl+(1:m)*h)';              % initial conditions
for j=1:n
  w(:,j+1)=a*w(:,j)+sigma*[lside(j);zeros(m-2,1);rside(j)];
end
w=[lside;w;rside];                  % attach boundary conds
x=(0:m+1)*h;t=(0:n)*k;
mesh(x,t,w')                        % 3-D plot of solution w
view(60,30);axis([xl xr yb yt -1 2])
```

```
function u=f(x)
% Use dot notation in functions f, l, and r
u=sin(2*pi*x).^2;

function u=l(t)
u=0*t;

function u=r(t)
u=0*t;
```

8.1.2 前向差分方法的稳定性分析

前面热方程模拟所呈现的奇怪性态已经把我们引入问题的核心. 在用前向差分方法解偏微分方程的过程中, 用切实可行的步长控制误差的放大是有效求解的关键因素.

恰如在以前研究的常微分方程情形中一样, 这种情形存在两类误差. 由于导数的近似表示, 离散化本身就产生截断误差. 从 Taylor 误差公式, 如 (8.4) 和 (8.5), 可知这些误差的大小. 此外, 方法本身还存在误差的放大. 为了研究这种放大, 我们需更仔细地观察用的是哪种有限差分方法.

冯 · 诺伊曼稳定性分析估量了误差的放大或扩大. 对一种稳定方法, 必须选取步长使误差的放大因子不大于 1.

在方程 (8.8) 中, 令 y_j 是满足 $\boldsymbol{y}_{j+1} = \boldsymbol{A}\boldsymbol{y}_j + \boldsymbol{s}_j$ 的精确解, 并且令 $\boldsymbol{w}_j$ 是计算得到的近似解, 它满足 $\boldsymbol{w}_{j+1} = \boldsymbol{A}\boldsymbol{w}_j + \boldsymbol{s}_j$. 它们的差 $\boldsymbol{e}_j = \boldsymbol{w}_j - \boldsymbol{y}_j$ 满足

$$
\begin{aligned}
\boldsymbol{e}_j &= \boldsymbol{w}_j - \boldsymbol{y}_j = \boldsymbol{A}\boldsymbol{w}_{j-1} + \boldsymbol{s}_{j-1} - (\boldsymbol{A}\boldsymbol{y}_{j-1} + \boldsymbol{s}_{j-1}) \\
&= \boldsymbol{A}(\boldsymbol{w}_{j-1} - \boldsymbol{y}_{j-1}) = \boldsymbol{A}\boldsymbol{e}_{j-1}
\end{aligned} \tag{8.9}
$$

附录 A 中的定理 A.7 表明, 为了保证误差 $\boldsymbol{e}_j$ 不放大, 我们要求 $\boldsymbol{A}$ 的谱半径 $\rho(\boldsymbol{A}) < 1$. 这个要求使得要对有限差分方法的步长 h 和 k 加以限制. 为了确定这些限制条件, 需要关于对称三对角矩阵的特征值的信息.

考虑下面的基本例子:

$$
\boldsymbol{T} = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ -1 & 1 & -1 & \ddots & \vdots \\ 0 & -1 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix}. \tag{8.10}
$$

定理 8.1 (8.10) 中的矩阵 $\boldsymbol{T}$ 的特征向量是 (8.12) 中的向量 $v_j, j = 1, \cdots, m$, 其相应的特征值 $\lambda_j = 1 - 2\cos \pi j/(m+1), j = 1, \cdots, m$.

证 首先回忆三角学中的正弦加法公式. 对任一整数 i 和实数 x, 可以把等式

$$\sin(i-1)x = \sin ix\cos x - \cos ix\sin x,$$
$$\sin(i+1)x = \sin ix\cos x + \cos ix\sin x,$$

相加得到

$$\sin(i-1)x + \sin(i+1)x = 2\sin ix\cos x,$$

它能写成

$$-\sin(i-1)x + \sin ix - \sin(i+1)x = (1-2\cos x)\sin ix. \tag{8.11}$$

等式 (8.11) 可以看成矩阵 $\boldsymbol{T}$ 的乘法. 固定整数 j, 定义向量

$$\boldsymbol{v}_j = \left[\sin\frac{j\pi}{m+1},\ \sin\frac{2\pi j}{m+1},\ \cdots,\ \sin\frac{m\pi j}{m+1}\right]. \tag{8.12}$$

注意: 其元素具有 $\sin ix$ 在 (8.11) 中的形式, 其中 $x = \pi j/(m+1)$. 现在可看到 (8.11) 意味着

$$\boldsymbol{T}\boldsymbol{v}_j = \left(1-2\cos\frac{\pi j}{m+1}\right)\boldsymbol{v}_j,\quad j=1,\cdots,m. \tag{8.13}$$

这里列出了 m 个特征向量和特征值.

当 j 从 $m+1$ 开始, 向量 $\boldsymbol{v}_j$ 重复出现. 因此, 如估计的那样, 恰好有 m 个特征向量 (见习题 6). $\boldsymbol{T}$ 的特征值都在 -1 和 3 之间.

定理 8.1 可以用来求主对角线和次对角线都是常数的任意对称, 三对角矩阵的特征值. 例如在 (8.8) 中的矩阵 $\boldsymbol{A}$ 能表示为 $\boldsymbol{A} = -\sigma\boldsymbol{T}+(1-\sigma)\boldsymbol{I}$. 根据定理 8.1, $\boldsymbol{A}$ 的特征值是 $-\sigma(1-2\cos\pi j/(m+1))+1-\sigma = 2\sigma(\cos\pi j/(m+1)-1)+1, j=1,\cdots,m$. 这里用到了这样的事实, 即当一个矩阵加上一个单位矩阵的若干倍之后, 它的特征值也加上这个倍数.

现在能使用定理 A.7 的准则. 由于对给定的自变量 $x=\pi j/(m+1), 1\leqslant j\leqslant m$, 有 $-2<\cos x-1<0$. 所以 $\boldsymbol{A}$ 的特征值可能在 $-4\sigma+1$ 和 1 之间. 假定扩散系数 $c>0$, 我们需限制 $\sigma<\frac{1}{2}$, 以保证 $\boldsymbol{A}$ 的特征值的绝对值小于 1, 即 $\rho(\boldsymbol{A})<1$.

可以把冯 • 诺伊曼稳定性分析的结果叙述如下:

定理 8.2 设 h 是空间步长, k 是时间步长, 当 $c>0$ 时, 把前向差分方法用于热方程 (8.2). 如果 $2ck<h^2$, 那么前向差分方法稳定.

我们的分析进一步证实了图 8-3 中的现象. 根据定义, 在图 8-3a 中, $\sigma=\frac{ck}{h^2}=\frac{(1)(0.004)}{(0.1)^2}=0.4<\frac{1}{2}$, 而在图 8-3b 中, $\sigma=\frac{(1)(0.005\ 5)}{(0.1)^2}=0.55>\frac{1}{2}$, 这里出现了误差放大. 因为显式差分方法的稳定性依赖于步长的选取, 所以称它为**条件稳定的**.

8.1.3 后向差分方法

作为供比较的选择方案, 通过使用隐式方法, 可以把有限差分方法重新做成具有较好的误差放大性质. 同前面一样, 在热方程中我们用中心差分公式代替 u_{xx}, 但是这次, 我们用后向差分公式

$$u_t = \frac{1}{k}(u(x,t) - u(x,t-k)) + \frac{k}{2}u_{tt}(x, x_0),$$

其中 $t-k < c_0 < t$, 来近似 u_t. 我们从第 6 章得到启发, 那里通过使用 (隐式) 后向 Euler 方法, 改进了显式 Euler 方法的稳定性, 而这种隐式 Euler 方法就使用了后向差分.

在点 (x_j, t_j) 处将差分公式代入到热方程, 有

$$\frac{1}{k}(w_{ij} - w_{i,j-1}) = \frac{c}{h^2}(w_{i+1,j} - 2w_{ij} + w_{i-1,j}), \tag{8.14}$$

它的局部截断误差是 $O(k) + O(h^2)$, 它与前向差分方法给出的误差相同. 式 (8.14) 可以重新写成

$$-\sigma w_{i+1,j} + (1+2\sigma)w_{ij} - \sigma w_{i-1,j} = w_{i,j-1},$$

其中 $\sigma = \frac{ck}{h^2}$, 并可表示成 $m \times m$ 矩阵方程

$$\begin{bmatrix} 1+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 1+2\sigma & -\sigma & \ddots & \vdots \\ 0 & -\sigma & 1+2\sigma & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 1+2\sigma \end{bmatrix} \begin{bmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{bmatrix} = \begin{bmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{bmatrix} + \sigma \begin{bmatrix} w_{0j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix} \tag{8.15}$$

这就是后向差分方法.

例 8.1 对热方程使用后向差分方法

$$\begin{cases} u_t = u_{xx}, & 0 \leqslant x \leqslant 1, \quad t \geqslant 0, \\ u(x,0) = \sin^2 2\pi x, & 0 \leqslant x \leqslant 1, \\ u(a,t) = 0, & t \geqslant 0, \\ u(b,t) = 0, & t \geqslant 0. \end{cases}$$

可以修改程序 8.1 以使用后向差分方法 (见计算机问题 3). 采用步长 $h = k = 0.1$, 得到图 8-4 所示的近似解. 与在图 8-3 中执行前向差分方法比较, 那里 $h = 0.1$, k 必须取得更小, 以避免不稳定. ◀

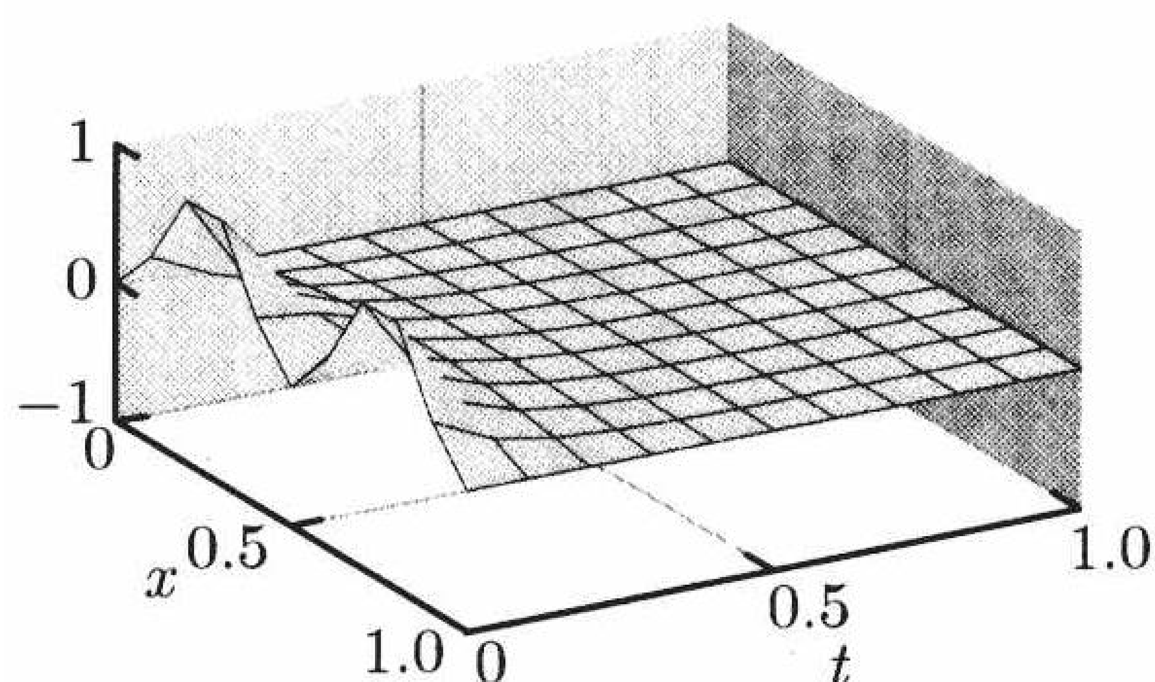


图 8-4 用后向差分方法近似解热方程 (8.2)：扩散系数 $c=1$, 步长 $h=k=0.1$

使隐式方法改进的理由是什么？后向差分方法的稳定性分析像显式情形一样进行. 后向差分方法 (8.15) 可以看成矩阵迭代

$$\boldsymbol{w}_j = \boldsymbol{A}^{-1}\boldsymbol{w}_{j-1} + \boldsymbol{b},$$

其中

$$\boldsymbol{A} = \begin{pmatrix} 1+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 1+2\sigma & -\sigma & \ddots & \vdots \\ 0 & -\sigma & 1+2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 1+2\sigma \end{pmatrix}. \tag{8.16}$$

和前向差分方法的冯 · 诺伊曼稳定性分析一样, 有关的量是 $\boldsymbol{A}^{-1}$ 的特征值. 因为 $\boldsymbol{A} = \sigma\boldsymbol{T} + (1+\sigma)\boldsymbol{I}$, 引理 8.1 指出 $\boldsymbol{A}$ 的特征值是

$$\sigma\left(1 - 2\cos\frac{\pi j}{m+1}\right) + 1 + \sigma = 1 + 2\sigma - 2\sigma\cos\frac{\pi j}{m+1},$$

而 $\boldsymbol{A}^{-1}$ 的特征值是其倒数. 为了保证 $\boldsymbol{A}^{-1}$ 的谱半径小于 1, 需要

$$|1 + 2\sigma(1 - \cos x)| > 1, \tag{8.17}$$

因为 $1 - \cos x > 0, \sigma = \frac{ck}{h^2} > 0$, 上式对任意 σ 都成立. 因此对任意 σ, 从而对任意选取的步长 h 和 k, 隐式方法稳定, 这样的稳定就定义为**无条件稳定** (unconditionally stable). 于是仅限于局部截断误差的考虑, 步长可取得大得多.

定理 8.3 *设 h 是空间步长, k 是时间步长, 当 $c > 0$ 时把后向差分方法用于热方程 (8.2). 对任意 h, k, 后向差分方法稳定.*

例 8.2 用后向差分方法解热方程

$$\begin{cases} u_t = 4u_{xx}, & 0 \leqslant x \leqslant 1, \quad 0 \leqslant t \leqslant 1, \\ u(x,0) = \mathrm{e}^{-x/2}, & 0 \leqslant x \leqslant 1, \\ u(0,t) = \mathrm{e}^{t}, & 0 \leqslant t \leqslant 1, \\ u(1,t) = \mathrm{e}^{t-\frac{1}{2}}, & 0 \leqslant t \leqslant 1. \end{cases}$$

可以验证其准确解是 $u(x,t)=e^{t-\frac{x}{2}}$. 令 $h=k=0.1$, $c=4$, 即 $\sigma=\frac{ck}{h^2}=40$. 矩阵 $\boldsymbol{A}$ 是 9×9, 而且在 10 个时间步的每一步都用 Gauss 消去法解 (8.15). 如图 8-5 所示.

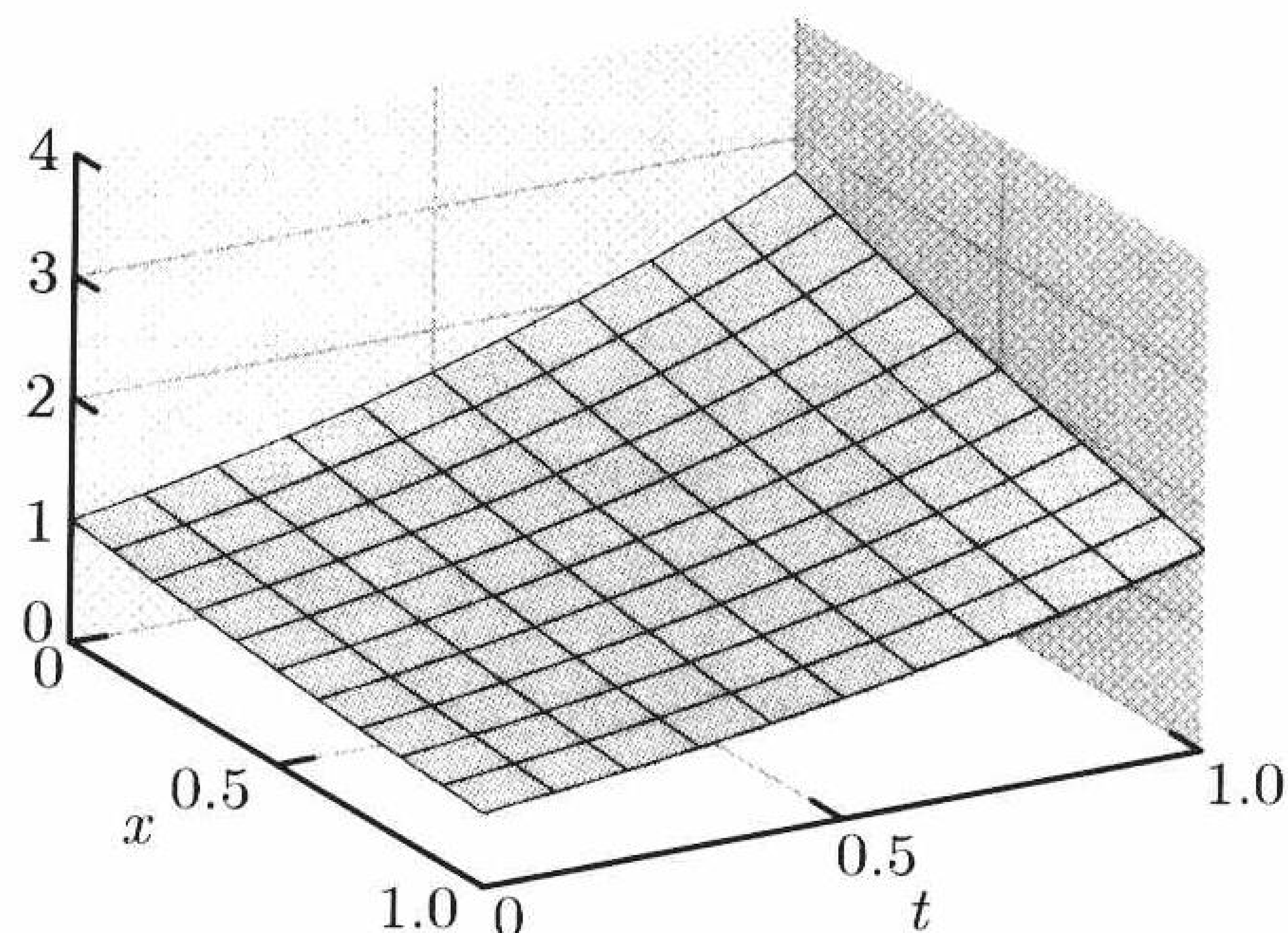


图 8-5　用后向差分方法解例 8.2 的近似解: 步长 $h=0.1$, $k=0.1$

因为对任意步长, 后向差分方法都稳定, 可以讨论由于在空间和时间方向的离散化而产生的截断误差的大小. 来自时间方向离散化的误差是 $O(k)$ 阶, 而来自空间方向离散化的误差是 $O(h^2)$ 阶. 这就意味着: 对于小步长 $h\approx k$, 来自时间步的误差将是主要的, 这是因为与 $O(k)$ 相比, $O(h^2)$ 可忽略不计. 换言之, 来自后向差分方法的误差可以粗略地描述为 $O(k)+O(h^2)\approx O(k)$.

为了说明这个结论, 我们用隐式有限差分方法, 对固定的 $h=0.1$ 和一系列递减的 k 去求例 8.2 的解. 表 8-1 说明在 $(x,t)=(0.5,1)$ 测得的误差随 k 线性地减小, 即当 k 减半时, 误差也减半. 如果 h 减小, 那么计算量将增加, 但是对给定的 k, 误差看上去却似乎相同.

表　8-1

h	k	$u(0.5,1)$	$w(0.5,1)$	误差
0.10	0.10	2.117 00	2.120 15	0.003 15
0.10	0.05	2.117 00	2.118 61	0.001 61
0.10	0.01	2.117 00	2.117 33	0.000 33

8.1.4　Crank-Nicolson 方法

到目前为止, 我们关于热方程的方法包括有时稳定的显式方法和永远稳定的隐式方法. 在稳定时它们都有 $O(k+h^2)$ 阶的误差. 为得到好的精度, 要求时间步长相当小.

Crank-Nicolson 方法是显式方法和隐式方法的结合, 它是无条件稳定的, 并且有误差阶 $O(h^2+k^2)$. 公式是稍微复杂一点, 但是由于提高了精度并保证了稳定性, 这些麻烦是值得的.

在热方程中, 用混合差分

$$\frac{1}{h^2}\left(\frac{1}{2}(w_{i+1,j}-2w_{ij}+w_{i-1,j})+\frac{1}{2}(w_{i+1,j-1}-2w_{i,j-1}+w_{i-1,j-1})\right)$$

代替 u_{xx}, 并用后向差分

$$\frac{1}{k}(w_{ij}-w_{i,j-1})$$

代替 u_t. 再令 $\sigma=\frac{ck}{h^2}$, 可以把热方程的近似表达式重新写成下列形式:

$$2w_{ij}-2w_{i,j-1}=\sigma\left[w_{i+1,j}-2w_{ij}+w_{i-1,j}+w_{i+1,j-1}-2w_{i,j-1}+w_{i-1,j-1}\right],$$

或者

$$-\sigma w_{i-1,j}+(2+2\sigma)w_{ij}-\sigma w_{i+1,j}=\sigma w_{i-1,j-1}+(2-2\sigma)w_{i,j-1}+\sigma w_{i+1,j-1}.$$

这就得出图 8-6 中所表出的模型.

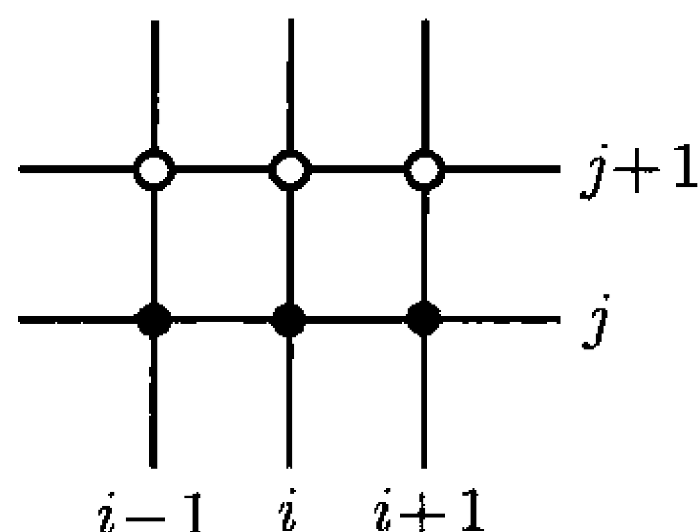


图 8-6 Crank-Nicolson 方法的网格点: 在每一时间步, 空心圆是未知量, 而实心圆是由前一步得到的

令 $\boldsymbol{w}_j=(w_{1j},\cdots,w_{mj})^{\mathrm{T}}$. **Crank-Nicolson 方法**的矩阵形式是

$$\boldsymbol{A}\boldsymbol{w}_j=\boldsymbol{B}\boldsymbol{w}_{j-1}+\sigma(\boldsymbol{s}_{j-1}+\boldsymbol{s}_j),$$

其中

$$\boldsymbol{A}=\begin{pmatrix} 2+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 2+2\sigma & -\sigma & \ddots & \vdots \\ 0 & -\sigma & 2+2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 2+2\sigma \end{pmatrix},$$

$$\boldsymbol{B}=\begin{pmatrix} 2-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 2-2\sigma & \sigma & \ddots & \vdots \\ 0 & \sigma & 2-2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma \\ 0 & \cdots & 0 & \sigma & 2-2\sigma \end{pmatrix},$$

$s_j = (w_{0j}, 0, \cdots, 0, w_{mj})^{\mathrm{T}}$. 对热方程应用 Crank-Nicolson 方法给出在图 8-7 中表出的结果, 这里的 $h = 0.1, k = 0.1$. 这个方法的 MATLAB 代码在程序 8.2 中给出.

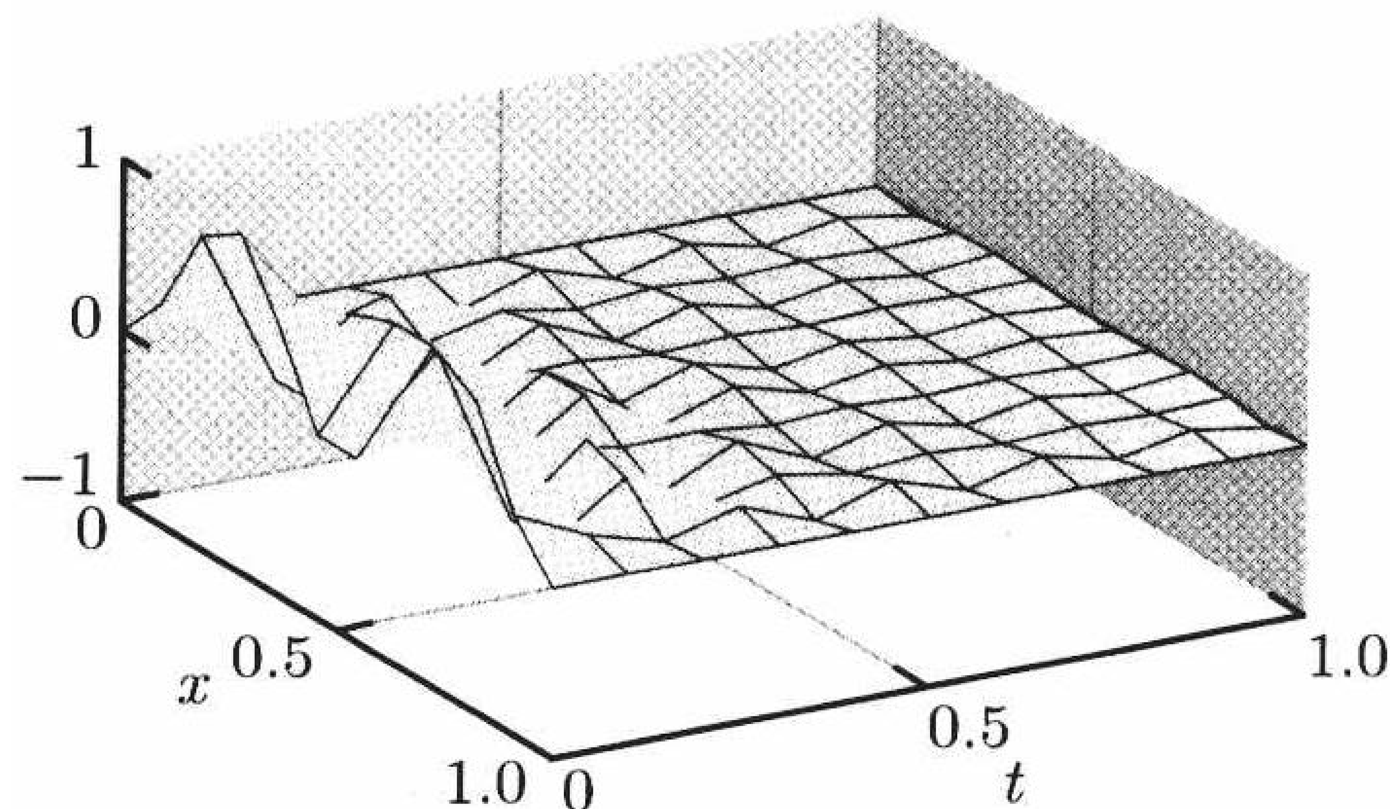


图 8-7 用 Crank-Nicolson 方法计算热方程 (8.2) 的近似解: 步长 $h = 0.1, k = 0.1$

```
% Program 8.2  Crank-Nicolson method
% input: space interval [xl,xr], time interval [yb,yt],
%        number of space steps M, number of time steps N
% output: solution w
% Example usage: w=crank(0,1,0,1,10,10)
function w=crank(xl,xr,yb,yt,M,N)
c=1;                                  % constant coefficient
h=(xr-xl)/M;k=(yt-yb)/N;              % step sizes
sigma=c*k/(h*h); m=M-1; n=N;
a=diag(2+2*sigma*ones(m,1))+diag(-sigma*ones(m-1,1),1);
a=a+diag(-sigma*ones(m-1,1),-1);      % define tridiagonal matrix a
b=diag(2-2*sigma*ones(m,1))+diag(sigma*ones(m-1,1),1);
b=b+diag(sigma*ones(m-1,1),-1);       % define tridiagonal matrix b
lside=l(yb+(0:n)*k); rside=r(yb+(0:n)*k);
w(:,1)=f(xl+(1:m)*h)';                % initial conditions
for j=1:n
    sides=[lside(j)+lside(j+1);zeros(m-2,1);rside(j)+rside(j+1)];
    w(:,j+1)=a\(b*w(:,j)+sigma*sides);
end
w=[lside;w;rside];
x=xl+(0:M)*h;t=yb+(0:N)*k;
mesh(x,t,w');
xlabel('x');ylabel('t');
axis([xl xr yb yt -1 2])

function u=f(x)
u=sin(2*pi*x).^2;
function u=l(t)
u=0*t;

function u=r(t)
u=0*t;
```

为了研究 Crank-Nicolson 方法的稳定性, 必须求出矩阵 $\boldsymbol{A}^{-1}\boldsymbol{B}$ 的谱半径, 其

中 $\boldsymbol{A}, \boldsymbol{B}$ 在前面已给定. 可以再次把问题中的矩阵重新表示成含 $\boldsymbol{T}$ 的项. 注意 $\boldsymbol{A} = \sigma\boldsymbol{T} + (2+\sigma)\boldsymbol{I}$, $\boldsymbol{B} = -\sigma\boldsymbol{T} + (2-\sigma)\boldsymbol{I}$. 把 $\boldsymbol{A}^{-1}\boldsymbol{B}$ 乘到 $\boldsymbol{T}$ 的第 j 个特征向量 $\boldsymbol{v}_j$, 得到

$$\begin{aligned}\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{v}_j &= (\sigma\boldsymbol{T} + (2+\sigma)\boldsymbol{I})^{-1}(-\sigma\lambda_j\boldsymbol{v}_j + (2-\sigma)\boldsymbol{v}_j) \\ &= \frac{1}{\sigma\lambda_j + 2 + \sigma}(-\sigma\lambda_j + 2 - \sigma)\boldsymbol{v}_j,\end{aligned}$$

这里 λ_j 是与 $\boldsymbol{v}_j$ 相关的 $\boldsymbol{T}$ 的特征值. $\boldsymbol{A}^{-1}\boldsymbol{B}$ 的特征值是

$$\frac{-\sigma\lambda_j + 2 - \sigma}{\sigma\lambda_j + 2 + \sigma} = \frac{4 - (\sigma(\lambda_j + 1) + 2)}{\sigma(\lambda_j + 1) + 2} = \frac{4}{L} - 1, \tag{8.18}$$

其中 $L = \sigma(\lambda_j + 1) + 2 > 2$, 这是因为 $\lambda_j > -1$. 因此, 特征值 (8.18) 在 -1 与 1 之间. 和隐式有限差分方法一样, Crank-Nicolson 方法是无条件稳定的.

定理 8.4 *当 $c > 0$ 时, 用于热方程 (8.2) 的 Crank-Nicolson 方法对任何步长 $h, k > 0$ 都是稳定的.*

作为本节的结束, 我们导出 Crank-Nicolson 方法的截断误差是 $O(h^2) + O(k^2)$. 加上它的无条件稳定性, 使它更优于热方程 $u_t = cu_{xx}$ 的前向和后向差分方法.

下面 4 个公式对导数是需要的. 我们假定, 如果需要, 解 u 的高阶导数是存在的. 根据习题 5.1.22, 有后向差分公式

$$u_t(x,t) = \frac{u(x,t) - u(x,t-k)}{k} + \frac{k}{2}u_{tt}(x,t) - \frac{k^2}{6}u_{ttt}(x,t_1), \tag{8.19}$$

其中 $t - k < t_1 < t$, 并假定偏导数都存在. 把 u_{xx} 展开成关于变量 t 的 Taylor 级数得

$$u_{xx}(x,t-k) = u_{xx}(x,t) - ku_{xxt}(x,t) - \frac{k^2}{2}u_{xxtt}(x,t_2),$$

其中 $t - k < t_2 < t$, 或者

$$u_{xx}(x,t) = u_{xx}(x,t-k) + ku_{xxt}(x,t) + \frac{k^2}{2}u_{xxtt}(x,t_2). \tag{8.20}$$

亮点　收敛性

由于 Crank-Nicolson 方法具有无条件稳定性 (定理 8.4) 和 (8.23) 表出的二阶收敛性, 所以对热方程来说, 它是一种优先选用的有限差分方法. 由于方程中的第一个偏导数 u_t, 它并不是直接导出的. 在本章后面讨论的波动方程和 Poisson 方程中, 仅出现二阶导数, 因此寻找稳定的二阶方法要容易得多.

二阶导数的中心差分公式给出

$$u_{xx}(x,t)=\frac{u(x+h,t)-2u(x,t)+u(x-h,t)}{h^2}+\frac{h^2}{12}u_{xxxx}(x_1,t), \tag{8.21}$$

$$u_{xx}(x,t-k)=\frac{u(x+h,t-k)-2u(x,t-k)+u(x-h,t-k)}{h^2}+\frac{h^2}{12}u_{xxxx}(x_2,t-k), \tag{8.22}$$

这里 x_1 和 x_2 在 x 和 $x+h$ 之间.

把上面 4 个等式代入热方程

$$u_t=c\left(\frac{1}{2}u_{xx}+\frac{1}{2}u_{xx}\right),$$

这里已把右端分裂成两项. 策略是用 (8.19) 代替左端, 用 (8.21) 代替右端的前一半, 用 (8.20) 和 (8.22) 代替右端的后一半. 结果是

$$\begin{aligned}&\frac{u(x,t)-u(x,t-k)}{k}+\frac{k}{2}u_{tt}(x,t)-\frac{k^2}{6}u_{ttt}(x,t_1)\\&=\frac{1}{2}c\left[\frac{u(x+h,t)-2u(x,t)+u(x-h,t)}{h^2}+\frac{h^2}{12}u_{xxxx}(x_1,t)\right]\\&+\frac{1}{2}c\left[ku_{xxt}(x,t)+\frac{k^2}{2}u_{xxtt}(x,t_2)\right.\\&\left.+\frac{u(x+h,t-k)-2u(x,t-k)+u(x-h,t-k)}{h^2}+\frac{h^2}{12}u_{xxxx}(x_2,t-k)\right].\end{aligned}$$

因此, 与使各差商相等有关的误差是余项

$$\begin{aligned}&-\frac{k}{2}u_{tt}(x,t)+\frac{k^2}{6}u_{ttt}(x,t_1)+\frac{ch^2}{24}[u_{xxxx}(x_1,t)+u_{xxxx}(x_2,t-k)]\\&+\frac{ck}{2}u_{xxt}(x,t)+\frac{ck^2}{4}u_{xxtt}(x,t_2).\end{aligned}$$

用 $u_t=cu_{xx}$ 就能简化上述表达式. 例如, 注意到 $cu_{xxt}=(cu_{xx})_t=u_{tt}$, 这就使误差表达式中的第一项和第四项消去. 截断误差为

$$\begin{aligned}&\frac{k^2}{6}u_{ttt}(x,t_1)+\frac{ck^2}{4}u_{xxtt}(x,t_2)+\frac{ch^2}{24}[u_{xxxx}(x_1,t)+u_{xxxx}(x_2,t-k)]\\&=\frac{k^2}{6}u_{ttt}(x,t_1)+\frac{k^2}{4}u_{ttt}(x,t_2)+\frac{h^2}{24c}[u_{tt}(x_1,t)+u_{tt}(x_2,t-k)].\end{aligned}$$

由关于变量 t 的 Taylor 展开得

$$u_{tt}(x_2,t-k)=u_{tt}(x_2,t)-ku_{ttt}(x_2,t_4),$$

使得截断误差等于

$$\frac{5}{12}k^2u_{ttt}(x,t_3)+\frac{h^2}{12c}u_{tt}(x_3,t)-\frac{h^2k}{24c}u_{ttt}(x_2,t_4)=O(h^2)+O(k^2)+\text{高阶项}. \tag{8.23}$$

我们断定求解热方程的 Crank-Nicolson 方法是一种二阶、无条件稳定的方法.

为了说明 Crank-Nicolson 方法的快速收敛性, 我们回到例 8.2 中的方程. 习题 5 和习题 6 也是探索收敛速度的.

例 8.3 应用 Crank-Nicolson 方法解热方程

$$\begin{cases} u_t = 4u_{xx}, & 0 \leqslant x \leqslant 1, \quad 0 \leqslant t \leqslant 1, \\ u(x,0) = \mathrm{e}^{-\frac{x}{2}}, & 0 \leqslant x \leqslant 1, \\ u(0,t) = \mathrm{e}^{t}, & 0 \leqslant t \leqslant 1, \\ u(1,t) = \mathrm{e}^{t-\frac{1}{2}}, & 0 \leqslant t \leqslant 1. \end{cases} \tag{8.24}$$

表 8-2 给出由前面计算所预测的 $O(h^2)+O(k^2)$ 的误差收敛性. 准确解 $u(x,t) = \mathrm{e}^{t-\frac{x}{2}}$ 在 $(x,t) = (0.5, 1)$ 处的值 $u = \mathrm{e}^{\frac{3}{4}}$. 注意当 h 和 k 减半时, 误差减小到原来的 $\frac{1}{4}$. 与例 8.2 在表 8-1 的误差进行比较:

表 8-2

h	k	$u(0.5,1)$	$w(0.5,1)$	误差
0.10	0.10	2.117 000 02	2.117 067 65	0.000 067 63
0.05	0.05	2.117 000 02	211 701 689	0.000 016 87
0.01	0.01	2117 000 02	2117 000 69	0.000 000 67

◀

习题 8.1

1. 证明函数 (a) $u(x,t) = \mathrm{e}^{2t+x} + \mathrm{e}^{2t-x}$ 和 (b) $u(x,t) = \mathrm{e}^{2t+x}$ 分别是热方程 $u_t = 2u_{xx}$ 在指定的初边值条件下的解.

$$\text{(a)} \begin{cases} u(x,0) = 2\cosh x, & 0 \leqslant x \leqslant 1, \\ u(0,t) = 2\mathrm{e}^{2t}, & 0 \leqslant t \leqslant 1, \\ u(1,t) = (\mathrm{e}^2+1)\mathrm{e}^{2t-1}, & 0 \leqslant t \leqslant 1; \end{cases} \qquad \text{(b)} \begin{cases} u(x,0) = \mathrm{e}^{x}, & 0 \leqslant x \leqslant 1, \\ u(0,t) = \mathrm{e}^{2t}, & 0 \leqslant t \leqslant 1, \\ u(1,t) = \mathrm{e}^{2t+1}, & 0 \leqslant t \leqslant 1. \end{cases}$$

2. 证明函数 (a) $u(x,t) = \mathrm{e}^{-\pi t}\sin \pi x$ 和 (b) $u(x,t) = \mathrm{e}^{-\pi t}\cos \pi x$ 分别是热方程 $\pi u_t = u_{xx}$ 在指定的初始边值条件下的解.

$$\text{(a)} \begin{cases} u(x,0) = \sin \pi x, & 0 \leqslant x \leqslant 1, \\ u(0,t) = 0, & 0 \leqslant t \leqslant 1, \\ u(1,t) = 0, & 0 \leqslant t \leqslant 1; \end{cases} \qquad \text{(b)} \begin{cases} u(x,0) = \cos \pi x, & 0 \leqslant x \leqslant 1, \\ u(0,t) = \mathrm{e}^{-\pi x}, & 0 \leqslant t \leqslant 1, \\ u(1,t) = -\mathrm{e}^{-\pi t}, & 0 \leqslant t \leqslant 1. \end{cases}$$

3. 假设 $f(x)$ 是 3 次多项式, 证明 $u(x,t) = f(x) + ctf''(x)$ 是初值问题 $u_t = cu_{xx}, u(x,0) = f(x)$ 的解.
4. 假设 $c < 0$, 对于热方程而言, 后向差分方法是无条件稳定吗? 请解释.
5. 证明特征向量方程 (8.13).
6. 证明: 对任意整数 m, (8.12) 中的非零向量 $\boldsymbol{v}_j$ 仅有 m 个不同的向量组成, 最多只改变符号.

计算机问题 8.1

1. 取步长 $h=0.1, k=0.002$, 用前向差分方法解热方程 $u_t=2u_{xx}, 0\leqslant x\leqslant 1, 0\leqslant t\leqslant 1$, 其初边值条件如下:

$$\text{(a)}\begin{cases} u(x,0)=2\cosh x, & 0\leqslant x\leqslant 1, \\ u(0,t)=2\mathrm{e}^{2t}, & 0\leqslant t\leqslant 1, \\ u(1,t)=(\mathrm{e}^2+1)\mathrm{e}^{2t-1}, & 0\leqslant t\leqslant 1; \end{cases} \quad \text{(b)}\begin{cases} u(x,0)=\mathrm{e}^x, & 0\leqslant x\leqslant 1, \\ u(0,t)=\mathrm{e}^{2t}, & 0\leqslant t\leqslant 1, \\ u(1,t)=\mathrm{e}^{2t+1}, & 0\leqslant t\leqslant 1. \end{cases}$$

用 MATLAB 的 **mesch** 命令画出近似解. 假如使用 $k>0.003$ 会发生什么现象? 与习题 1 中的准确解进行比较.

2. 考虑方程 $\pi u_t=u_{xx}, 0\leqslant x\leqslant 1, 0\leqslant t\leqslant 1$, 以及下述两种初边值条件. 取步长 $h=0.1$, 那么步长 k 取什么值时, 前向差分方法稳定? 使用步长 $h=0.1, k=0.01$ 的前向差分方法并与习题 2 中的准确解比较:

$$\text{(a)}\begin{cases} u(x,0)=\sin\pi x, & 0\leqslant x\leqslant 1, \\ u(0,t)=0, & 0\leqslant t\leqslant 1, \\ u(1,t)=0, & 0\leqslant t\leqslant 1; \end{cases} \quad \text{(b)}\begin{cases} u(x,0)=\cos\pi x, & 0\leqslant x\leqslant 1, \\ u(0,t)=\mathrm{e}^{-\pi t}, & 0\leqslant t\leqslant 1, \\ u(1,t)=-\mathrm{e}^{-\pi t}, & 0\leqslant t\leqslant 1. \end{cases}$$

3. 用后向差分方法解计算机问题 1. 在 $(x,t)=(0.5,1)$ 处作一张准确值、近似值及误差的表, 取步长 $h=0.02$ 及 $k=0.02, 0.01, 0.005$.
4. 用后向差分方法解计算机问题 2. 在 $(x,t)=(0.3,1)$ 处作一张准确值、近似值及误差的表, 取步长 $h=0.1$ 及 $k=0.02, 0.01, 0.005$.
5. 用 Crank-Nicolson 方法解计算机问题 1, 在 $(x,t)=(0.5,1)$ 处作一张准确值、近似值及误差的表, 取长步 $h=k=0.02, 0.01, 0.005$.
6. 用 Crank-Nicolson 方法解计算机问题 2, 在 $(x,t)=(0.3,1)$ 处作一张准确值、近似值及误差的表, 取步长 $h=k=0.1, 0.05, 0.025$.

8.2 双曲型方程

双曲型方程对显式方法设置的严格约束较少. 本节针对称为波动方程的这种有代表性的双曲型方程, 来探索有限差分方法的稳定性. 我们将引入 CFL 条件, 它一般是偏微分方程解法稳定的必要条件.

8.2.1 波动方程

考虑偏微分方程

$$u_{tt}=c^2u_{xx}, \tag{8.25}$$

$a\leqslant x\leqslant b, t\geqslant 0$. 与标准形式 (8.1) 相比, 我们计算 $AC-B^2=-c^2<0$, 因此这个方程是双曲型的. 这个例子称为波动速度为 c 的**波动方程**. 保证存在唯一解的典型的初边值条件是

$$\begin{cases} u(x,0) = f(x), & a \leqslant x \leqslant b, \\ u_t(x,0) = g(x), & a \leqslant x \leqslant b, \\ u(a,t) = l(t), & t \geqslant 0, \\ u(b,t) = r(t), & t \geqslant 0. \end{cases} \tag{8.26}$$

与热方程例子相比, 由于方程中的高阶时间导数, 需要额外的初始条件. 直观地说, 波动方程描述了随时间进展的沿着 x 方向的波的传播. 为了明确所发生的事情, 我们需要知道波在每一点的初始形状和初始速度.

波动方程是很广泛的一类自然现象的模型, 它包括从太阳大气层中的磁波到小提琴弦的振动. 对于小提琴而言, 方程中涉及的振幅 u 表示弦的物理位移; 对于空气中的声波移动, u 表示局部气压.

我们将把有限差分法用到波动方程 (8.25), 并分析它的稳定性. 恰如在抛物型情形中一样, 有限差分方法在如同图 8-1 那样的网格上运行. 网格点是 $(x_i, t_j), x_i = a + ih, t_j = jk$, 其中 h 和 k 是步长. 和前面一样, 我们将用 w_{ij} 表示 $u(x_i, t_j)$ 的近似值.

为了离散波动方程, 在 x 方向和 t 方向用中心差分公式 (8.4) 代替二阶偏导数

$$\frac{w_{i,j+1} - 2w_{ij} + w_{i,j-1}}{k^2} - c^2 \frac{w_{i-1,j} - 2w_{ij} + w_{i+1,j}}{h^2} = 0.$$

令 $\sigma = \dfrac{ck}{h}$, 可以求得下一时间步的解, 并且把上面的离散化方程写为

$$w_{i,j+1} = (2 - 2\sigma^2) w_{ij} + \sigma^2 w_{i-1,j} + \sigma^2 w_{i+1,j} - w_{i,j-1}. \tag{8.27}$$

由于需要前两步 ($j-1$ 和 j) 时间上的值, 所以在第一时间步不能用公式 (8.27). 这一点类似于与起始多步常微分方程方法有关的问题. 为了解这个问题, 可以引入三点中心差分公式去近似解 u 的一阶时间导数:

$$u_t(x_i, t_j) \approx \frac{w_{i,j+1} - w_{i,j-1}}{2k},$$

代入第一时间步 (x_i, t_1) 的初始数据, 得到

$$g(x_i) = u_t(x_i, t_0) \approx \frac{w_{i1} - w_{i,-1}}{2k},$$

或者

$$w_{i,-1} \approx w_{i1} - 2kg(x_i). \tag{8.28}$$

把 (8.28) 代入 $j = 0$ 时的有限差分公式 (8.27), 给出

$$w_{i1} = (2 - 2\sigma^2) w_{i0} + \sigma^2 w_{i-1,0} + \sigma^2 w_{i+1,0} - w_{i1} + 2kg(x_i),$$

它可以解出 w_{i1}:

$$w_{i1}=(1-\sigma^2)w_{i0}+kg(x_i)+\frac{\sigma^2}{2}(w_{i-1,0}+w_{i+1,0}). \tag{8.29}$$

公式 (8.29) 用于第一时间步. 这是使初始速度信息 g 加入计算的一种方法. 公式 (8.27) 可用于所有以后的时间步. 由于对空间导数和时间导数用了二阶公式, 这个有限差分方法的误差将是 $O(h^2)+O(k^2)$ (见计算机问题 3 和计算机问题 4).

为了把这个有限差分方法写成矩阵形式, 定义

$$\boldsymbol{A}=\begin{pmatrix} 2-2\sigma^2 & \sigma^2 & 0 & \cdots & 0 \\ \sigma^2 & 2-2\sigma^2 & \sigma^2 & \ddots & \vdots \\ 0 & \sigma^2 & 2-2\sigma^2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma^2 \\ 0 & \cdots & 0 & \sigma^2 & 2-2\sigma^2 \end{pmatrix}. \tag{8.30}$$

初始方程 (8.29) 可以写成

$$\begin{pmatrix} w_{11} \\ \vdots \\ w_{m1} \end{pmatrix}=\frac{1}{2}\boldsymbol{A}\begin{pmatrix} w_{10} \\ \vdots \\ w_{m0} \end{pmatrix}+k\begin{pmatrix} g(x_1) \\ \vdots \\ g(x_m) \end{pmatrix}+\frac{1}{2}\sigma^2\begin{pmatrix} w_{00} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,0} \end{pmatrix},$$

(8.27) 的后续各步可以由下式给出:

$$\begin{pmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{pmatrix}=\boldsymbol{A}\begin{pmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{pmatrix}-\begin{pmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{pmatrix}+\sigma^2\begin{pmatrix} w_{0j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{pmatrix}.$$

代入其他的初边值条件, 这两个方程写成

$$\begin{pmatrix} w_{11} \\ \vdots \\ w_{m1} \end{pmatrix}=\frac{1}{2}\boldsymbol{A}\begin{pmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{pmatrix}+k\begin{pmatrix} g(x_1) \\ \vdots \\ g(x_m) \end{pmatrix}+\frac{1}{2}\sigma^2\begin{pmatrix} l(t_0) \\ 0 \\ \vdots \\ 0 \\ r(t_0) \end{pmatrix},$$

(8.27) 的后续各步可由下式给出：

$$\begin{pmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{pmatrix} = \boldsymbol{A} \begin{pmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{pmatrix} - \begin{pmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{pmatrix} + \sigma^2 \begin{pmatrix} l(t_j) \\ 0 \\ \vdots \\ 0 \\ r(t_j) \end{pmatrix}. \tag{8.31}$$

例 8.4 当波速 $c = 2$, 初始条件 $f(x) = \sin \pi x$, $g(x) = l(x) = r(2) = 0$ 时, 用显式有限差分方法解波动方程.

图 8-8 给出在 $c = 2$ 时波动方程的近似解. 显式有限差分方法是条件稳定的, 必须小心地选取步长, 以避免解法的不稳定性. 图 8-8a 表示稳定的选取是 $h = 0.05$ 及 $k = 0.025$, 而图 8-8b 表示不稳定的选取是 $h = 0.05$ 及 $k = 0.032$. 当时间步长 k 相对于空间步长 h 太大时, 用于波动方程的显式有限差分方法是不稳定的. ◀

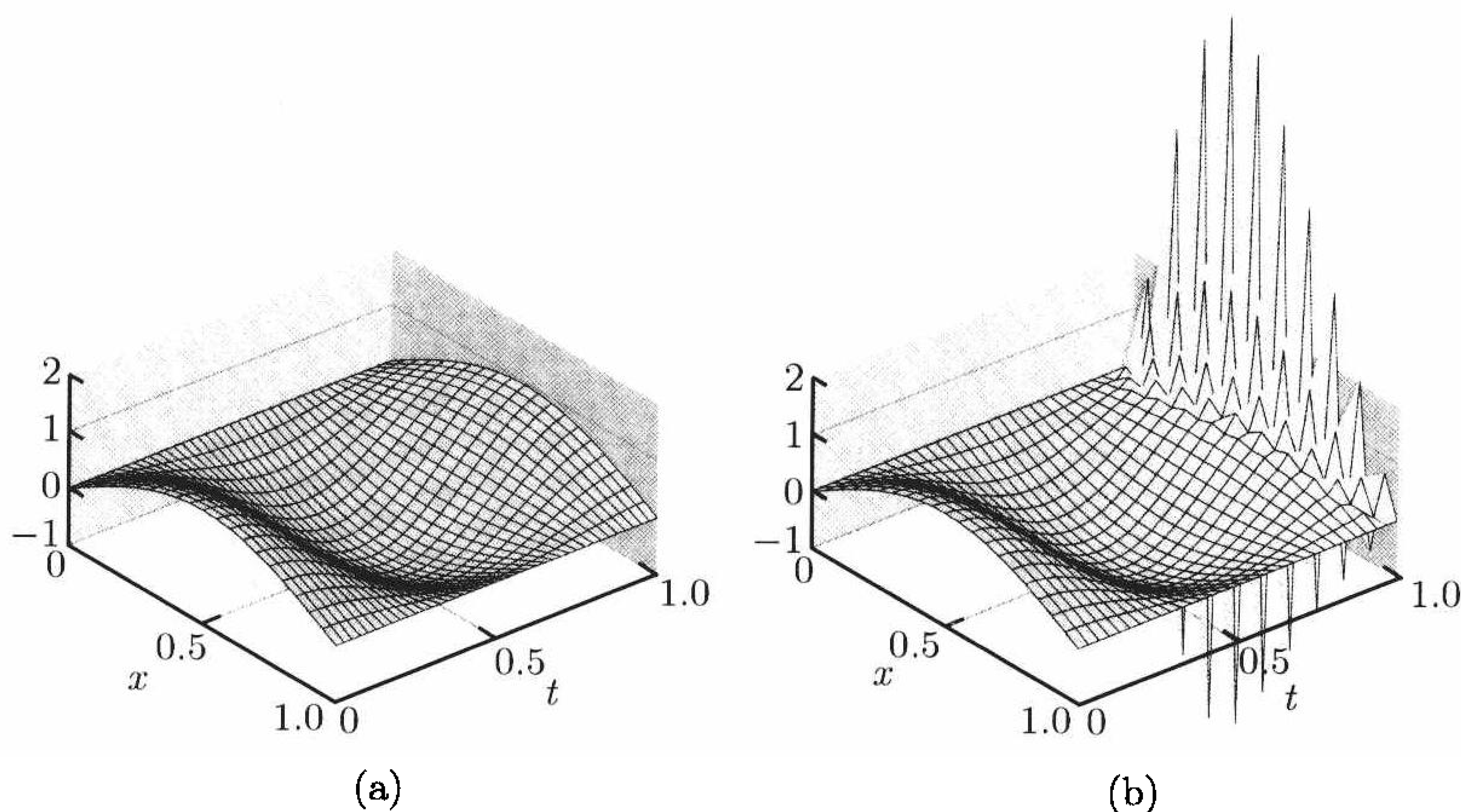


图 8-8 用显式有限差分方法求解例 8.4 中的波动方程. 空间步长是 $h = 0.05$. (a) 对时间步长 $k = 0.025$, 方法稳定; (b) 对 $k = 0.032$, 方法不稳定

8.2.2 CFL 条件

矩阵形式方便了我们分析用于波动方程的显式有限差分方法的稳定性特征. 如定理 8.5 中所叙述的分析结果, 解释了图 8-8.

定理 8.5 *当波动速度 $c > 0$ 时, 如果 $\sigma = \frac{ck}{h} < 1$, 那么用于波动方程的有限差分方法稳定.*

证 方程 (8.31) 的向量形式是

$$\boldsymbol{w}_{j+1} = \boldsymbol{A}\boldsymbol{w}_j - \boldsymbol{w}_{j-1} + \sigma^2 \boldsymbol{s}_j, \tag{8.32}$$

这里 $\boldsymbol{s}_j$ 使边界条件满足. 由于 $\boldsymbol{w}_{j+1}$ 依赖于 $\boldsymbol{w}_j$ 和 $\boldsymbol{w}_{j-1}$, 为了研究误差的放大性, 我们把 (8.32) 重新写成

$$\begin{pmatrix} \boldsymbol{w}_{j+1} \\ \boldsymbol{w}_j \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_j \\ \boldsymbol{w}_{j-1} \end{pmatrix} + \sigma^2 \begin{pmatrix} \boldsymbol{s}_j \\ \boldsymbol{0} \end{pmatrix}, \tag{8.33}$$

并把这个方法看成一步递推. 只要分块矩阵

$$\boldsymbol{A}' = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}$$

的特征值的绝对值不大于 1, 那么误差将不会放大.

令 $\lambda \neq 0, (y, z)^{\mathrm{T}}$ 是 $\boldsymbol{A}'$ 的特征值/特征向量对, 因此,

$$\lambda \boldsymbol{y} = \boldsymbol{A}\boldsymbol{y} - \boldsymbol{z},$$

$$\lambda \boldsymbol{z} = \boldsymbol{y},$$

它意味着

$$\boldsymbol{A}\boldsymbol{y} = \left(\frac{1}{\lambda} + \lambda\right) \boldsymbol{y},$$

所以 $\mu = \frac{1}{\lambda} + \lambda$ 是 $\boldsymbol{A}$ 的特征值. $\boldsymbol{A}$ 的特征值在 $2 - 4\sigma^2$ 和 2 之间 (见习题 5). 假设 $\sigma \leqslant 1$, 这就意味着 $-2 \leqslant \mu \leqslant 2$. 要完成证明, 只需对复数 λ 证明, $\frac{1}{\lambda} + \lambda$ 是实数且最大绝对值是 2, 就意味着 $|\lambda| = 1$ (见习题 6).

我们遵循 R. Courant, K. Friedrichs and H. Levy[5], 把 $\frac{ck}{h}$ 称为方法的 **CFL 数**. 一般地, 为了使偏微分方程的解法稳定, CFL 数必须不大于 1. 因为 c 是波速, 这就意味着解在一个时间步所移动的距离 ck 应该不超过空间步 h. 图 8-8a 和图 8-8b 分别说明 CFL 数为 1 和 1.28 的情形. 约束 $ck \leqslant h$ 称为波动方程的 **CFL 条件**.

定理 8.5 说明, 对于波动方程, CFL 条件就意味着有限差分方法的稳定性. 对于更一般的双曲型方程, CFL 条件对稳定性是必要的, 但并不都是充分的. 详细论述见 [14].

在波动方程中的波动速度参数 c 控制波的传播速度. 图 8-9 表明, 对于 $c = 6$, 在一个时间单位内, 正弦波初始条件振荡 3 次, 相当于 $c = 2$ 的情形的 3 倍.

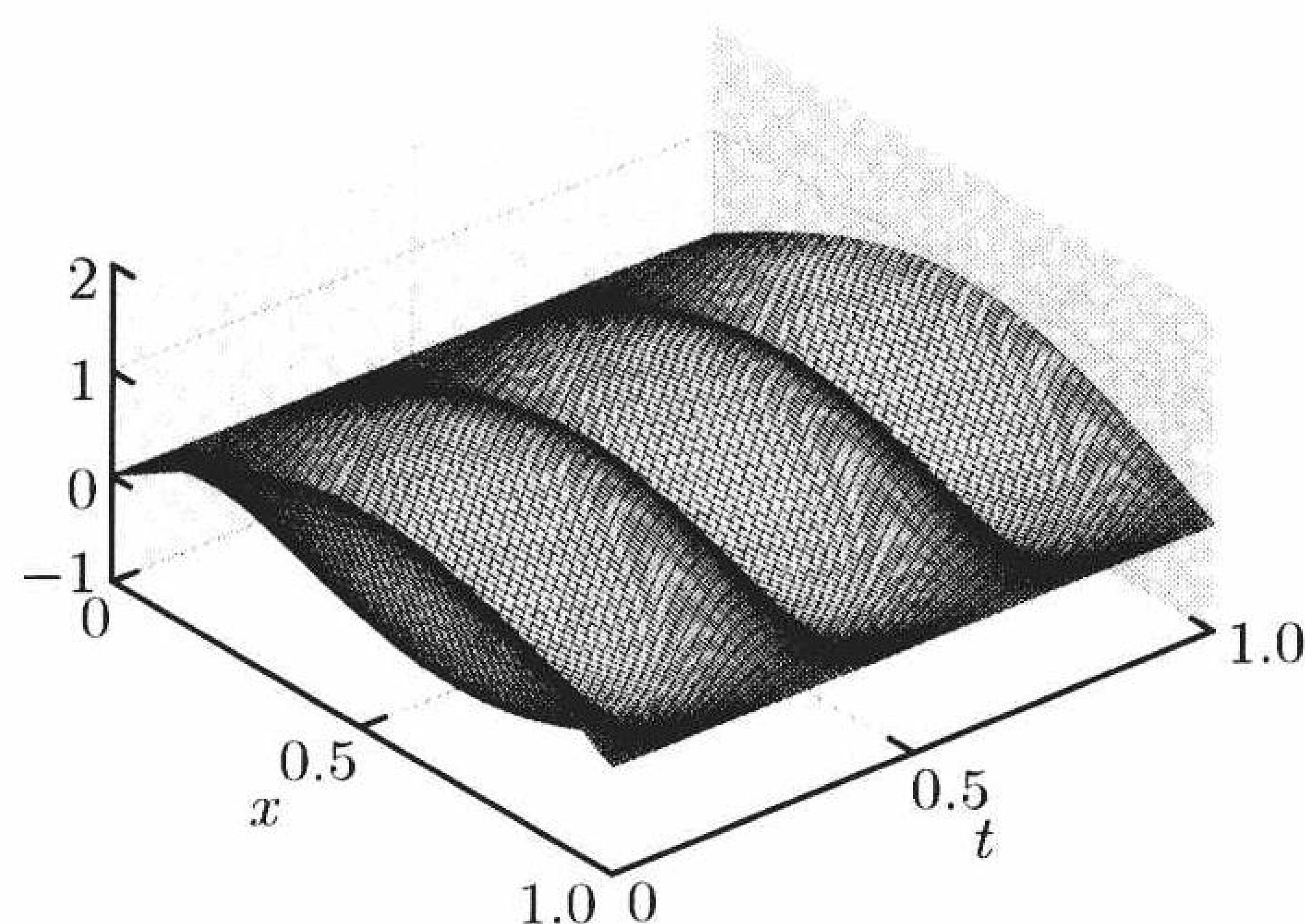


图 8-9 用于 $c=6$ 的波动方程的显式有限差分方法: 步长 $h=0.05, k=0.008$ 满足 CFL 条件

习题 8.2

1. 证明函数 (a) $u(x,t)=\sin \pi x\cos 4\pi t$, (b) $u(x,t)=\mathrm{e}^{-x-2t}$, (c) $u(x,t)=\ln(1+x+t)$ 分别是带有以下给定初边值条件的波动方程的解.

$$\text{(a)}\begin{cases} u_{tt}=16u_{xx}, & \\ u(x,0)=\sin \pi x, & 0\leqslant x\leqslant 1, \\ u_t(x,0)=0, & 0\leqslant x\leqslant 1, \\ u(0,t)=0, & 0\leqslant t\leqslant 1, \\ u(1,t)=0, & 0\leqslant t\leqslant 1; \end{cases}\qquad \text{(b)}\begin{cases} u_{tt}=4u_{xx}, & \\ u(x,0)=\mathrm{e}^{-x}, & 0\leqslant x\leqslant 1, \\ u_t(x,0)=-2\mathrm{e}^{-x}, & 0\leqslant x\leqslant 1, \\ u(0,t)=\mathrm{e}^{-2t}, & 0\leqslant t\leqslant 1, \\ u(1,t)=\mathrm{e}^{-1-2t}, & 0\leqslant t\leqslant 1; \end{cases}$$

$$\text{(c)}\begin{cases} u_{tt}=u_{xx}, & \\ u(x,0)=\ln(1+x), & 0\leqslant x\leqslant 1, \\ u_t(x,0)=\frac{1}{1+x}, & 0\leqslant x\leqslant 1, \\ u(0,t)=\ln(1+t), & 0\leqslant t\leqslant 1, \\ u(1,t)=\ln(2+t), & 0\leqslant t\leqslant 1. \end{cases}$$

2. 证明函数 (a) $u(x,t)=\sin \pi x\sin 2\pi t$, (b) $u(x,t)=(x+2t)^5$, (c) $u(x,t)=\sinh x\cosh 2t$ 分别是带有以下给定初边值条件的波动方程的解:

$$\text{(a)}\begin{cases} u_{tt}=4u_{xx}, & \\ u(x,0)=0, & 0\leqslant x\leqslant 1, \\ u_t(x,0)=2\pi\sin \pi x, & 0\leqslant x\leqslant 1, \\ u(0,t)=0, & 0\leqslant t\leqslant 1, \\ u(1,t)=0, & 0\leqslant t\leqslant 1; \end{cases}\qquad \text{(b)}\begin{cases} u_{tt}=4u_{xx}, & \\ u(x,0)=x^5, & 0\leqslant x\leqslant 1, \\ u_t(x,0)=10x^4, & 0\leqslant x\leqslant 1, \\ u(0,t)=32t^5, & 0\leqslant t\leqslant 1, \\ u(1,t)=(1+2t)^5, & 0\leqslant t\leqslant 1; \end{cases}$$

$$\text{(c)}\begin{cases} u_{tt}=4u_{xx}, & \\ u(x,0)=\sinh x, & 0\leqslant x\leqslant 1, \\ u_t(x,0)=0, & 0\leqslant x\leqslant 1, \\ u(0,t)=0, & 0\leqslant t\leqslant 1, \\ u(1,t)=\frac{1}{2}\left(\mathrm{e}-\frac{1}{\mathrm{e}}\right)\cosh 2t, & 0\leqslant t\leqslant 1. \end{cases}$$

3. 证明 $u_1(x,t)=\sin \alpha x\cos c\alpha t$ 和 $u_2(x,t)=\mathrm{e}^{x+ct}$ 都是波动方程 (8.25) 的解.

4. 证明: 如果 $s(x)$ 二次可微, 那么 $u(x,t)=s(\alpha x+c\alpha t)$ 是波动方程 (8.25) 的解.

5. 证明 (8.30) 中 $\boldsymbol{A}$ 的特征值在 $2-4\sigma^2$ 和 2 之间.

6. 设 λ 是复数. (a) 证明: 假如 $\lambda+\frac{1}{\lambda}$ 是实数, 那么 $|\lambda|=1$ 或者 λ 是实数. (b) 证明: 如果 λ 是实数并且 $|\lambda+\frac{1}{\lambda}|\leqslant 2$, 那么 $|\lambda|=1$.

计算机问题 8.2

1. 取 $h=0.05, k=\frac{h}{c}$, 在 $0\leqslant x\leqslant 1, 0\leqslant t\leqslant 1$ 上, 用有限差分方法解习题 1 中的初边值问题. 用 MATLAB 的 mesh 命令绘制这个解.

2. 取 $h=0.05$ 以及 k 足够小以满足 CFL 条件, 在 $0\leqslant x\leqslant 1, 0\leqslant t\leqslant 1$ 上, 用有限差分方法解习题 2 中的初边值问题. 绘制这个解.

3. 对习题 1 中的波动方程, 把在 $(x,t)=(\frac{1}{4},\frac{3}{4})$ 处的近似解及其误差当作步长 $h=ck=2^{-p}(p=4,\cdots,8)$ 的函数, 列出上述近似解及其误差的表.

4. 对习题 2 中的波动方程, 把在 $(x,t)=(\frac{1}{4},\frac{3}{4})$ 处的近似解及其误差当作步长 $h=ck=2^{-p}(p=4,\cdots,8)$ 的函数, 列出上述近似解及其误差的表.

8.3　椭圆型方程

前两节处理了与时间有关的方程. 扩散方程把热流建模为时间的函数, 而波动方程探索波的运动. 本节的核心是椭圆型方程, 用来模拟定常状态. 例如, 边界是给定温度的平面区域上的热的定常分布, 能用椭圆型方程来模拟. 由于在椭圆型方程中时间通常不是一个因子, 我们用 x 和 y 表示自变量.

定义 8.6　*设 $u(x,y)$ 是二次可微函数, 定义 u 的* **Laplace 算子**为

$$\Delta u=u_{xx}+u_{yy}.$$

对连续函数 $f(x,y)$, 偏微分方程

$$\Delta u(x,y)=f(x,y) \tag{8.34}$$

称为 **Poisson 方程**. *当 $f(x,y)=0$ 时, Poisson 方程称为* **Laplace 方程**. *Laplace 方程的解称为***调和函数**.

为了与标准形式 (8.1) 比较, 我们计算 $AC-B^2>0$, 因此 Poisson 方程是椭圆型的. 保证唯一解而给定的额外条件一般是边界条件. 有两种常见的边界条件可用. Dirichlet 边界条件在区域 R 的边界 ∂R 上给定解 $u(x,y)$ 的值. 诺伊曼边界条件在边界上给定方向导数 $\frac{\partial u}{\partial n}$ 的值, 这里 n 表示外向单位法向量.

例 8.5　证明 $u(x,y)=x^2-y^2$ 是在 $[0,1]\times[0,1]$ 上给定 Dirichlet 边界条件

$$u(x,0)=x^2,\quad u(x,1)=x^2-1,$$
$$u(0,y)=-y^2,\quad u(1,y)=1-y^2$$

的 Laplace 方程的解.

Laplace 算子是 $\Delta u = u_{xx} + u_{yy} = 2 - 2 = 0$. 列出的边界条件分别针对单位正方形的底边、顶边、左边和右边, 通过代入容易得到检验. ◀

Poisson 方程和 Laplace 方程在经典物理中普遍存在, 这是因为它们的解代表势能. 例如, 电场强度 E 是静电势 u 的梯度, 或者

$$E = -\nabla u.$$

反过来, 由 Maxwell 方程

$$\nabla E = \frac{\rho}{\varepsilon}$$

可知, 电场强度与电荷密度 ρ 有关, 这里 ε 是介电常数. 把这两个方程结合起来得到

$$\Delta u = \nabla(\nabla u) = \frac{-\rho}{\varepsilon},$$

即电势 u 的 Poisson 方程. 在零电荷密度这种特殊情形下, 电势满足 Laplace 方程 $\Delta u = 0$.

势能的许多其他情形可以用 Poisson 方程来模拟. 空气动力学中低速空气薄片, 称为不可压缩无旋流, 就是 Laplace 方程的解. 由质量密度 ρ 分布产生的重力势能 u 满足 Poisson 方程

$$\Delta u = 4\pi G\rho,$$

其中 G 表示引力常量. 稳定状态的热分布, 譬如像当 $t \to \infty$ 时热方程解的极限, 是由 Poisson 方程来模拟的. 在实例检验 8 中, 一种不同的 Poisson 方程用来模拟冷却片上的热分布.

余下的章节介绍解椭圆方程的两种方法. 第一种是紧随抛物型和双曲型方程建立起来的有限差分方法. 第二种是在第 7 章中形成的解边值问题的有限元方法.

8.3.1 椭圆型方程的有限差分方法

假定在平面的矩形区域 $[x_l, x_r] \times [y_b, y_t]$ 上给出 Dirichlet 边界条件. 令 M 和 N 分别是 x 方向和 y 方向的网格步数. $h = \frac{x_r - x_l}{M}, k = \frac{y_t - y_b}{N}$ 是 x 方向和 y 方向的步长. 如图 8-10a 所示, 在矩形网格点上解这个方程. 由于边界条件在边界上是已知的, 因此解需要在 mn 个点上计算, 这里 $m = M - 1, n = N - 1$. 图 8-10b 表示对相同的解值的另一个编号系统, 这些解将在这个方法中用到, 其中我们设

$$v_{i+(j-1)m} = u_{ij}. \tag{8.35}$$

有限差分方法包括用差商近似导数. Laplace 算子中的二阶导数都能用中心差分公式 (8.4). Poisson 方程 $\Delta u = f$ 具有有限差分形式

$$\frac{u(x-h,y) - 2u(x,y) + u(x+h,y)}{h^2} + \frac{u(x,y-k) - 2u(x,y) + u(x,y+k)}{k^2} = f(x,y).$$

定义 $r \equiv \frac{h^2}{k^2}$, 可以把上式写成

$$u(x-h,y)+u(x+h,y)-2(1+r)u(x,y)+r\left[u(x,y-k)+u(x,y+k)\right]=h^2f(x,y) \tag{8.36}$$

它的离散化形式是

$$u_{i-1,j}+u_{i+1,j}-2(1+r)u_{ij}+r(u_{i,j-1}+u_{i,j+1})=h^2f(x_i,y_j), \tag{8.37}$$

这里 $x_i=x_l+ih, y_j=y_b+jk$. 按图 8-10b 中另一种表示法, 上式就对应于

$$v_{i-1+(j-1)m}+v_{i+1+(j-1)m}-2(1+r)v_{i+(j-1)m}+r(v_{i+(j-2)m}+v_{i+jm})=h^2f(x_i,y_j). \tag{8.38}$$

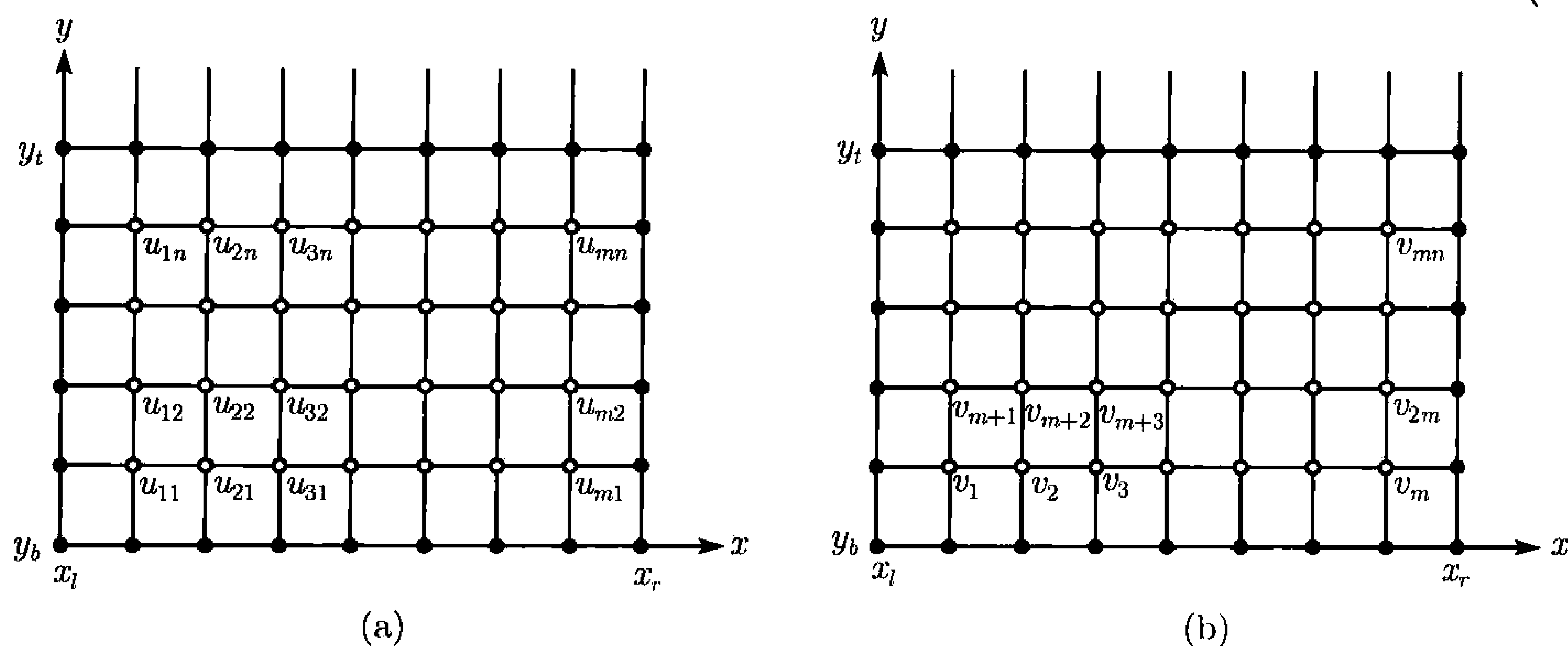


图 8-10 用于解带有 Dirichlet 边界条件的 Poisson 方程的有限差分方法用的网格: (a) 空心圆是未知值, 实心圆是给定的边界数据; (b) 适合线性方程组逐行穿过的网格点顺序的编号系统 (8.35)

它有助于把网格分成格外要特别对待的 3 类, 取决于它们的水平和垂直方向上的相邻网格点.

内核 (inner core) 点, $u_{ij}, 1<i<m, 1<j<n$, 它的 4 个邻点全部在网格区域内部.

边界 (outer ring) 点 u_{ij} 刚好对应以下之一: $i=1, i=m, j=1, j=n$. 这些点有一个邻点在边界上.

四角 (four corner) 点, $u_{11}, u_{1n}, u_{m1}, u_{mn}$, 它们有两个邻点在边界上.

解矩阵方程 $\boldsymbol{A}\boldsymbol{v}=\boldsymbol{b}$ 确定 $\boldsymbol{v}_k$, 在内核结点, (8.38) 表示矩阵 $\boldsymbol{A}$ 的第 $i+(j-1)m$ 行是

$$[\texttt{zeros(1,i}-1+(\texttt{j}-2)\texttt{m)}\ r\ \texttt{zeros(1,m}-2)\ 1$$
$$-2(1+r)\ 1\ \texttt{zeros(1,m}-2)\ r\ \texttt{zeros(1,(n}-\texttt{j)m}-\texttt{i)}],$$

这里 MATLAB 记号 zeros(1, p) 用来表示 p 个连续零. 边界点上的值由 Dirichlet 条件给出, 当需要边界结点和四角结点上的值时, 就代入 (8.37).

最后, 这些方程构成了一个线性方程组, 可以用第 2 章中适当的方法求解它. 我们用图 8-10 的计算系统来确定网格点的次序, 即在每行从左到右移动. 在这种假设下, 结构矩阵 $\boldsymbol{A}$ 和荷载矩阵 $\boldsymbol{b}$ 的每一行就能直接填入, 如例 8.6 所示.

例 8.6 取 $M = N = 4$, 用有限差分方法求在 $[0,1] \times [1,2]$ 上的 Laplace 方程的近似解, Dirichlet 边界条件如下:

$$u(x,1) = \ln(x^2+1), \quad u(x,2) = \ln(x^2+4),$$
$$u(0,y) = 2\ln y, \quad u(1,y) = \ln(y^2+1).$$ ◀

我们将在正方形的 9 个网格结点上用正确解 $u(x,y) = \ln(x^2+y^2)$ 与近似解进行比较. 因为 $M = N = 4$, 所以网格步长 $h = k = \frac{1}{4}$, $r = \frac{h^2}{k^2} = 1$. 唯一的内核点的值是 $u_{22} = u\left(\frac{1}{2}, \frac{3}{2}\right)$, 它相应于 v_5. 根据 (8.37), 它的方程是

$$u_{12} + u_{32} - 2(1+r)u_{22} + r(u_{21} + u_{23}) = 0. \tag{8.39}$$

在另一种编号系统下, 这个方程是

$$v_4 + v_6 - 2(1+r)v_5 + rv_2 + rv_8 = 0,$$

9×9 结构矩阵 $\boldsymbol{A}$ 的相应行是

$$[0 \quad r \quad 0 \quad 1 \quad -2(1+r) \quad 1 \quad 0 \quad r \quad 0].$$

4 个边界点上的值 $u_{21}, u_{12}, u_{32}, u_{23}$, 分别相应于 v_2, v_4, v_6, v_8, 并且每一个都取方程的一个边界值. 这些方程是

$$\begin{aligned}
&u_{11} + u_{31} - 2(1+r)u_{21} + r(u_{20} + u_{22}) = 0 \qquad (i=2, j=1),\\
&u_{02} + u_{22} - 2(1+r)u_{12} + r(u_{11} + u_{13}) = 0 \qquad (i=1, j=2),\\
&u_{22} + u_{42} - 2(1+r)u_{32} + r(u_{31} + u_{33}) = 0 \qquad (i=3, j=2),\\
&u_{13} + u_{33} - 2(1+r)u_{23} + r(u_{22} + u_{24}) = 0 \qquad (i=2, j=3),
\end{aligned} \tag{8.40}$$

其中边界条件是

$$u_{20} = u\left(\frac{1}{2}, 1\right) = \ln\frac{5}{4}, \quad u_{02} = u\left(0, \frac{3}{2}\right) = \ln\frac{9}{4},$$
$$u_{42} = u\left(1, \frac{3}{2}\right) = \ln\frac{13}{4}, \quad u_{24} = u\left(\frac{1}{2}, 2\right) = \ln\frac{17}{4}.$$

4 个角点处的值 $u_{11}, u_{31}, u_{13}, u_{33}$ 各取两个边界值. u_{11} 的方程是

$$u_{21} - 4u_{11} + u_{12} = -u_{01} - u_{10} = -2\ln\frac{5}{4} - \ln\left(\left(\frac{1}{4}\right)^2 + 1\right), \tag{8.41}$$

这里的右端全由已知的边界值填满. 结合 (8.39), (8.40) 及 (8.41), 得知要解出 u_{ij} 的矩阵方程是

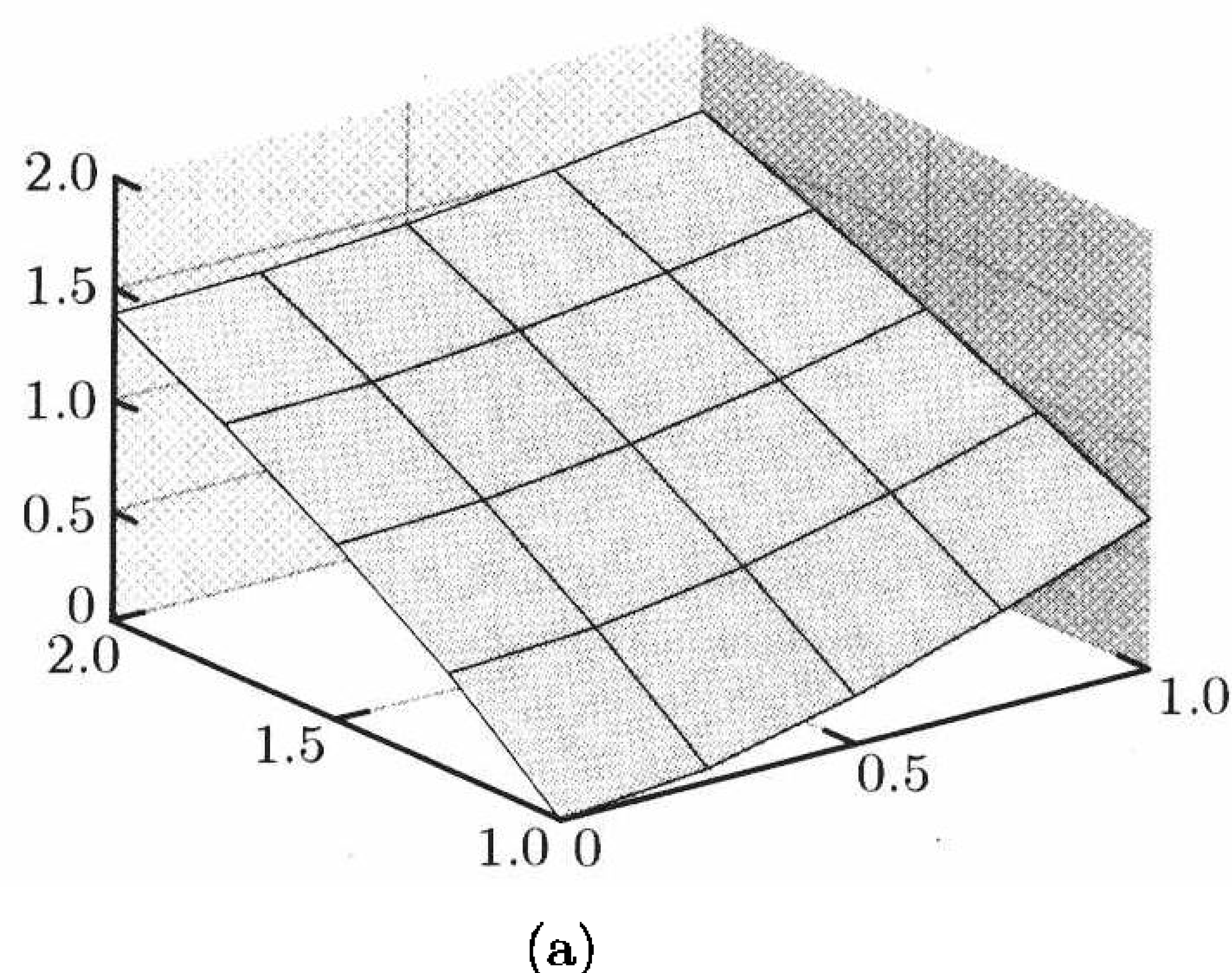
$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{21} \\ u_{31} \\ u_{12} \\ u_{22} \\ u_{32} \\ u_{13} \\ u_{23} \\ u_{33} \end{pmatrix} = \begin{pmatrix} -0.506\,9 \\ -0.223\,1 \\ -1.387\,3 \\ -0.810\,9 \\ 0 \\ -1.178\,7 \\ -2.521\,0 \\ -1.446\,9 \\ -2.919\,7 \end{pmatrix}, \tag{8.42}$$

并且解由以下 9 个 u 值组成:

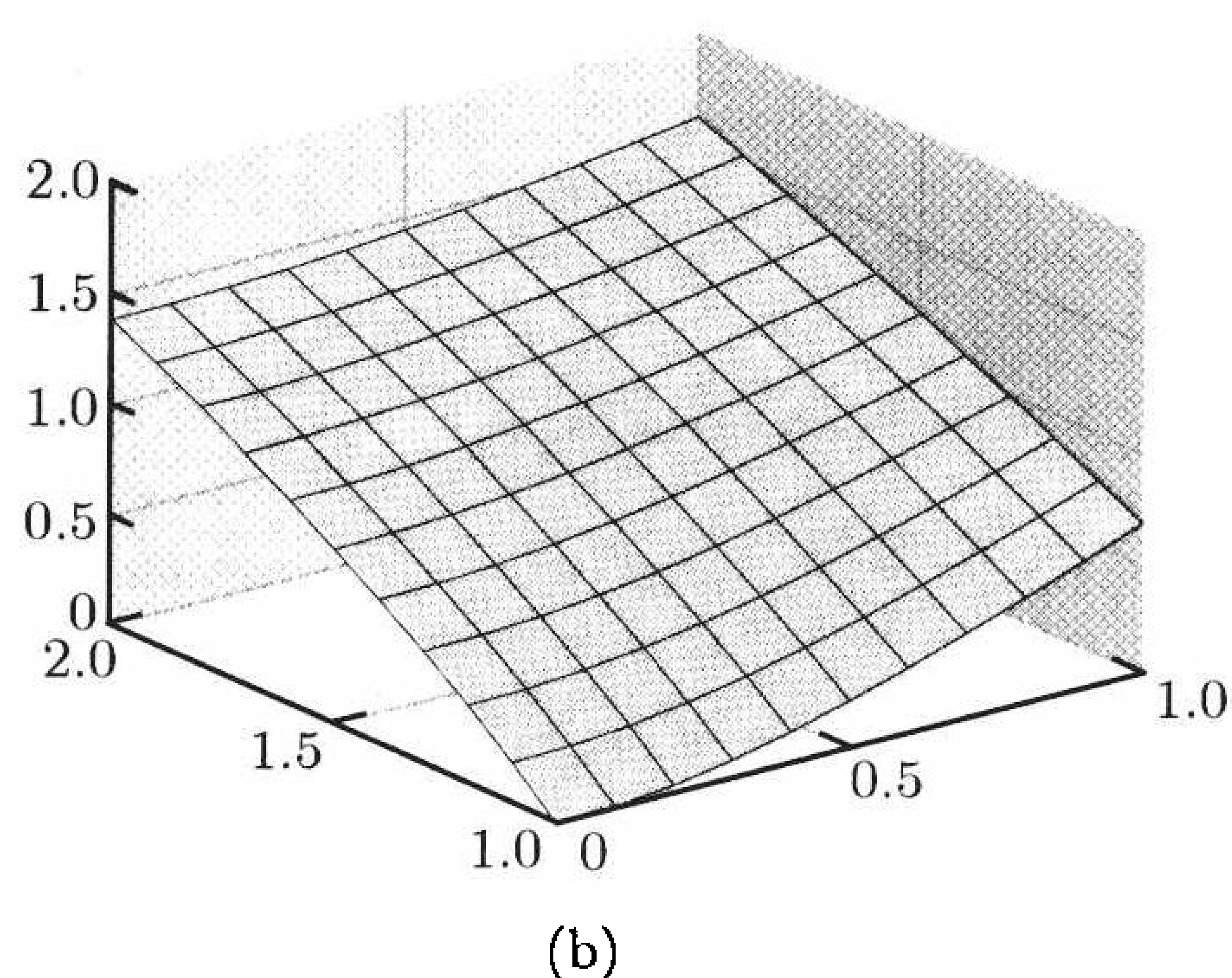
$$\begin{aligned} &u_{13} = 1.139\,0, \quad && u_{23} = 1.197\,4, \quad && u_{33} = 1.287\,8, \\ &u_{12} = 0.837\,6, && u_{22} = 0.915\,9, && u_{32} = 1.034\,1, \\ &u_{11} = 0.484\,7, && u_{21} = 0.594\,4, && u_{31} = 0.753\,9. \end{aligned}$$

图 8-11a 绘出了近似解 u_{ij}. 在相同的点, 它与准确解 $u(x,y) = \ln(x^2+y^2)$ 比较接近:

$$\begin{aligned} &u\left(\tfrac{1}{4},\tfrac{7}{4}\right) = 1.139\,4, \quad && u\left(\tfrac{2}{4},\tfrac{7}{4}\right) = 1.197\,7, \quad && u\left(\tfrac{3}{4},\tfrac{7}{4}\right) = 1.287\,9, \\ &u\left(\tfrac{1}{4},\tfrac{6}{4}\right) = 0.838\,3, && u\left(\tfrac{2}{4},\tfrac{6}{4}\right) = 0.916\,3, && u\left(\tfrac{3}{4},\tfrac{6}{4}\right) = 1.034\,1, \\ &u\left(\tfrac{1}{4},\tfrac{5}{4}\right) = 0.485\,5, && u\left(\tfrac{2}{4},\tfrac{5}{4}\right) = 0.594\,7, && u\left(\tfrac{3}{4},\tfrac{5}{4}\right) = 0.753\,8. \end{aligned}$$ ◀



(a)



(b)

图 8-11　有限差分方法解例 8.6 的椭圆型偏微分方程: (a) $M = N = 4$, 步长 $h = k = 0.25$; (b) $M = N = 10$, 步长 $h = k = 0.1$

有限差分方法的 MATLAB 代码如下:

```
% Program 8.3 Finite difference solver for 2D Poisson equation
%   with Dirichlet boundary conditions on a rectangle
% Input: rectangle domain [xl,xr]x[yb,yt], covered by MxN grid
% Output: matrix w holding solution values on MxN grid
function w=poisson(xl,xr,yb,yt,M,N)
m=M-1;n=N-1;
h=(xr-xl)/M;h2=h^2;k=(yt-yb)/N;
r=h2/k^2;s=2*(1+r);
x=xl+(xr-xl)*(0:M)/M;              % set mesh values
y=yb+(yt-yb)*(0:N)/N;
z=zeros(1,m-2);
a=zeros(m*n,m*n);b=zeros(m*n,1);  % initialize structure matrix
% load structure matrix a
% inner core
for i=2:m-1
  for j=2:n-1
    a(i+(j-1)*m,:)=[zeros(1,i-1+(j-2)*m) r z 1 -s 1 z r ...
     zeros(1,(n-j)*m-i)];
    b(i+(j-1)*m)=h2*f(x(i+1),y(j+1));
  end
end
% outer ring
j=1;                          % bottom row
for i=2:m-1
  a(i+(j-1)*m,:)=[zeros(1,i-2) 1 -s 1 z r zeros(1,(n-j)*m-i)];
  b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-r*gbottom(x(i+1));
end
j=n;                          % top row
for i=2:m-1
  a(i+(j-1)*m,:)=[zeros(1,i-1+(j-2)*m) r z 1 -s 1 zeros(1,m-i-1)];
  b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-r*gtop(x(i+1));
end
i=1;                          % left side
for j=2:n-1
  a(i+(j-1)*m,:)=[zeros(1,i-1+(j-2)*m) r z 0 -s 1 z r ...
   zeros(1,(n-j)*m-i)];
  b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-gleft(y(j+1));
end
i=m;                          % right side
for j=2:n-1
  a(i+(j-1)*m,:)=[zeros(1,(j-1)*m-1) r z 1 -s 0 z r ...
   zeros(1,(n-j)*m-i)];
  b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-gright(y(j+1));
end
% four corners
i=1;j=1;                      % bottom left
a(i+(j-1)*m,:)=[-s 1 z r zeros(1,(n-1)*m-1)];
b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-r*gbottom(x(i+1))-gleft(y(j+1));
i=m;j=1;                      % bottom right
```

```
a(i+(j-1)*m,:)=[z 1 -s 0 z r zeros(1,(n-2)*m)];
b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-r*gbottom(x(i+1))-gright(y(j+1));
i=1;j=n;                      % top left
a(i+(j-1)*m,:)=[zeros(1,(n-2)*m) r z 0 -s 1 zeros(1,m-2)];
b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-r*gtop(x(i+1))-gleft(y(j+1));
i=m;j=n;                      % top right
a(i+(j-1)*m,:)=[zeros(1,(n-1)*m-1) r z 1 -s];
b(i+(j-1)*m)=h2*f(x(i+1),y(j+1))-r*gtop(x(i+1))-gright(y(j+1));
v=a\b;                           % solve for solution
w=zeros(m,n);
for i=1:m                        % put solution into mesh
  for j=1:n
    w(i,j)=v(i+(j-1)*m);
  end
end
w1=[gbottom(x(2:M))' w gtop(x(2:M))']; % add boundary conditions
w1=[gleft(y);w1;gright(y)];
mesh(x,y,w1')
function u=f(x,y)                % right hand side of equation
u=0;

function u=gbottom(x)            % bottom of rectangle
% Use dot notation
u=log(x.^2+1);

function u=gtop(x)               % top of rectangle
u=log(x.^2+4);

function u=gleft(y)              % left side of rectangle
u=2*log(y);

function u=gright(y)             % right side of rectangle
u=log(y.^2+1);
```

因为用了二阶有限差分公式, 有限差分方法 `Poisson.m` 的误差对于 h 和 k 是二阶的. 图 8-11b 对于 $h = k = 0.1$ 表示出了更为精确的近似解. 可对矩形区域写出 MATLAB 代码 `Poisson.m`, 但对更一般的区域须作改变.

再举一个用 Laplace 方程计算势能的例子.

例 8.7 在正方形 $[0,1] \times [0,1]$ 上求静电势, 假定内部无电荷并满足以下边界条件:

$$u(x,0) = \sin \pi x, \quad u(x,1) = \sin \pi x,$$
$$u(0,y) = 0, \quad u(1,y) = 0.$$

势能 u 满足带 Dirichlet 边界条件的 Laplace 方程. 在 `Poisson.m` 中取步长 $h = k = 0.1$ 或 $M = N = 10$, 结果由图 8-12 给出. ◀

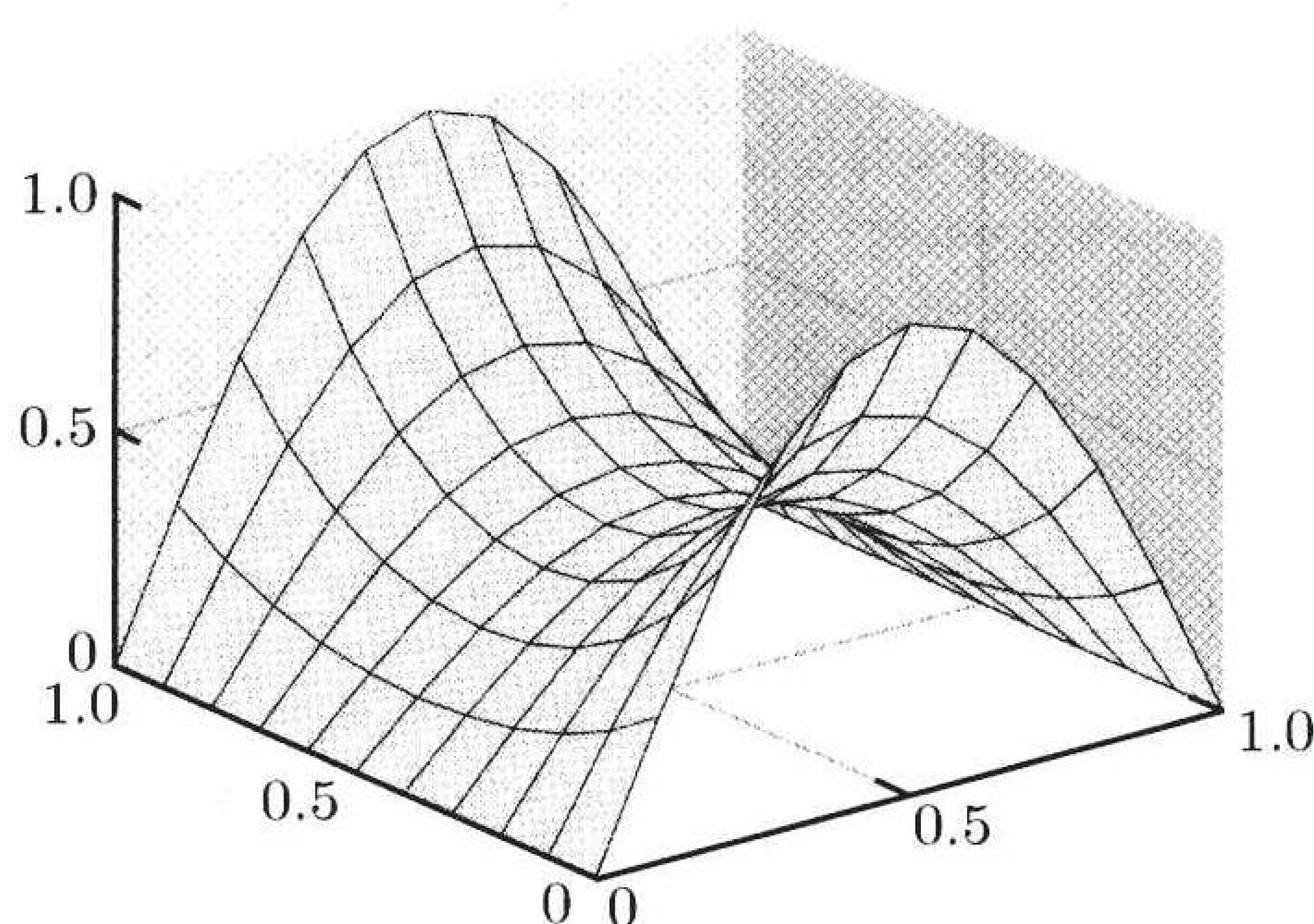


图 8-12 Laplace 方程中的静电势, 边界条件由例 8.7 给出

实例检验 8 冷却片上的热分布

散热器用来把过多的热量从来源处移开. 在这个主题中, 我们将对沿着一个散热器矩形片的稳态分布建立模型. 热能将沿着其中一侧进入这个 (冷却) 片. 主要目的是设计这个片的尺寸以便使温度保持在安全允许限度内.

这个片的形状是一个薄的矩形平板, 它的尺寸为 $L_x \times L_y$, 厚度是 δ 毫米. 为了使问题简化, 我们将用 $u(x, y)$ 来表示温度, 并且认为在第三维沿着 δ 毫米, 它是常数.

热运动按下面 3 种方式进行: 传导、对流和辐射. 传导是指相邻分子间能量的传送, 可能是由于电子的运动, 而在对流中是分子本身在运动. 这里我们将不考虑辐射这种借助于光子的能量运动.

传导通过传导物质进行, 服从 Fourier 第一定律

$$q = -KA\nabla u, \tag{8.43}$$

其中 q 是每单位时间的热能 [单位: watts (瓦特)], A 是该物质的横截面面积, ∇u 是温度的梯度. 常数 K 称为该物质的**热传导系数** (thermal conductivity). 对流遵从牛顿冷却定律

$$q = -HA(u - u_b), \tag{8.44}$$

其中 H 是一个比例常数, 称为**对流换热系数** (convective heat transfer coefficient). u_b 是周围流体 (本例中是空气) 的外界温度或**整体温度** (bulk temperature).

这个冷却片是 z 方向为 δ 毫米的 $[0, L_x] \times [0, L_y]$ 矩形, 如图 8-13a 所示. 在冷却片内部的 $\Delta x \times \Delta y \times \delta$ 盒子内部, 能量均衡地沿着 x 和 y 轴进入冷却片, 即每单位时间进入盒子的能量等于流出的能量. 由于传导, 热流经过 $\Delta y \times \delta$ 两面及 $\Delta x \times \delta$ 两面进入盒子, 而由于对流, 热流经过 $\Delta x \times \Delta y$ 两面进入盒子, 这样就产生了稳态方程:

$$\begin{aligned} &- K\Delta y\delta u_x(x, y) + K\Delta y\delta u_x(x + \Delta x, y) - K\Delta x\delta u_y(x, y) \\ &+ K\Delta x\delta u_y(x, y + \Delta y) - 2H\Delta x\Delta y u(x, y) = 0. \end{aligned} \tag{8.45}$$

这里为了方便, 取外界温度 $u_b = 0$, 这样 u 将表示冷却片和周围的温度差.

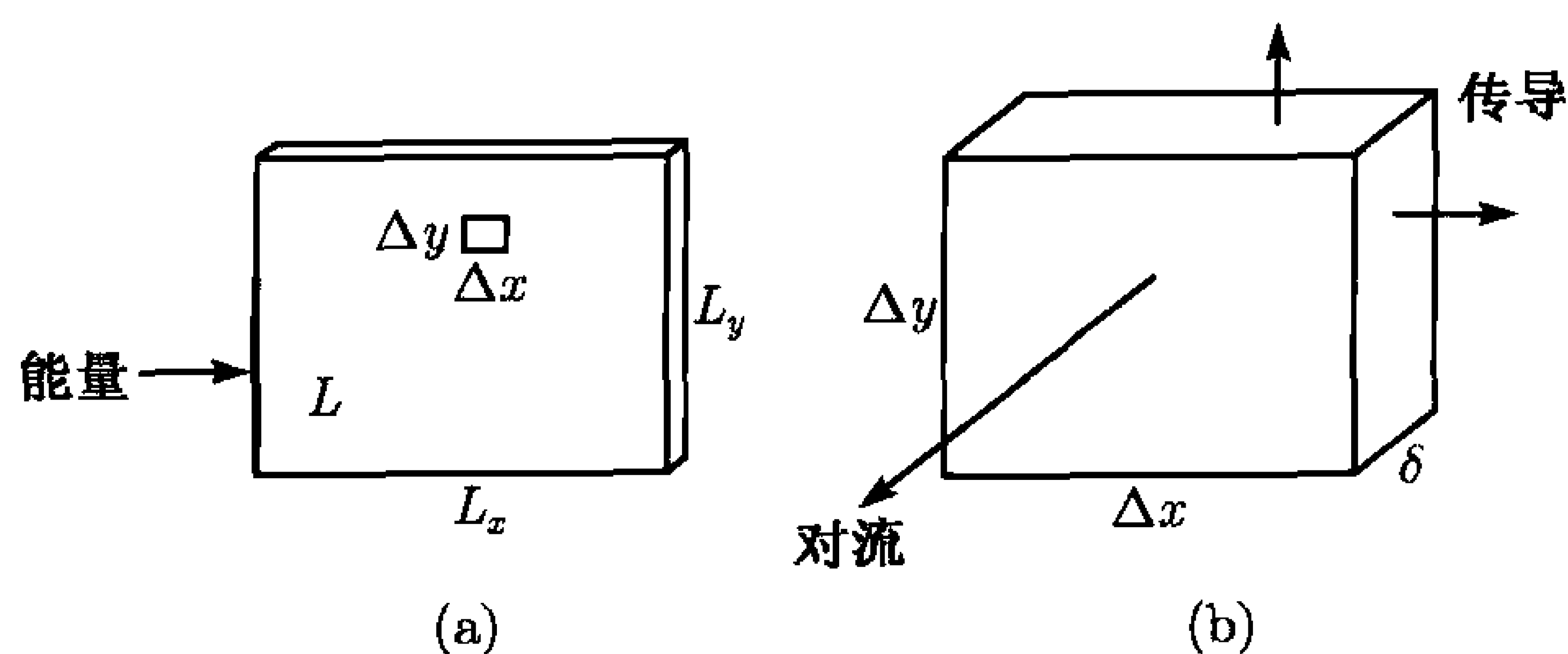


图 8-13 实例检验 8 中的冷却片. (a) 能量输入在冷却片的左侧沿着区间 $[0, L]$ 出现. (b) 内部小盒子的能量转移是通过沿着 x 方向和 y 方向的传导以及沿着空气界面的对流进行的

除以 $\Delta x \Delta y$ 就得到

$$K\delta \frac{u_x(x+\Delta x, y) - u_x(x,y)}{\Delta x} + K\delta \frac{u_y(x, y+\Delta y) - u_y(x,y)}{\Delta y} = 2Hu(x,y),$$

当 $\Delta x \to 0, \Delta y \to 0$ 取极限, 结果就是椭圆型偏微分方程

$$u_{xx} + u_{yy} = \frac{2H}{K\delta} u. \tag{8.46}$$

类似的论证得到沿垂直边的对流或 **Robin** 边界条件

$$Ku_x + Hu = 0,$$

以及沿水平边的 Robin 边界条件

$$Ku_y + Hu = 0.$$

能量沿着冷却片的左侧进入, 它遵从 Fourier 定律

$$\delta K u_x = -\frac{P}{L},$$

其中 P 是总能量, L 是输入长度.

在步长分别为 h 和 k 的离散网格上, 有限差分近似能用于把偏微分方程 (8.46) 近似为

$$\frac{u_{i+1,j} - 2u_{ij} - u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{k^2} = \frac{2H}{K\delta} u_{ij}.$$

这个离散形式用于内点 (x_i, y_j), 其中 $0 < i < M, 0 < j < N$, 这里 M, N 是整数. 冷却片的边界服从 Robin 条件, 可用一阶导数的近似公式

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + O(h^2).$$

把它用到 4 条边上, 就得到

$$\frac{-3u_{i0} + 4u_{i1} - u_{i2}}{2k} = -\frac{H}{K} u_{i0} \qquad \text{(底边)},$$

$$\frac{u_{i,N-2} - 4u_{i,N-1} + 3u_{iN}}{2k} = -\frac{H}{K} u_{iN} \qquad \text{(顶边)},$$

$$\frac{-3u_{0j} + 4u_{1j} - u_{2j}}{2h} = -\frac{H}{K} u_{0j} \qquad \text{(左边)},$$

$$\frac{u_{M-2,j} - 4u_{M-1,j} + 3u_{M_j}}{2h} = -\frac{H}{K} u_{M_j} \qquad \text{(右边)},$$

能量沿着左侧进入, 可用方程

$$\frac{-3u_{0j}+4u_{1j}-u_{2j}}{2h}=-\frac{P}{LK\delta}, \tag{8.47}$$

需要解含 $(M+1)(N+1)$ 个未知数 $u_{ij}(0\leqslant i\leqslant M, 0\leqslant j\leqslant n)$ 的 $(M+1)(N+1)$ 个方程, 从而得到偏微分方程的近似解.

假定冷却片是铝构成的, 铝的导热率 $K=1.68\text{W}/(\text{cm}\cdot{}^\circ\text{C})$, 再假定对流换热系数 $H=0.005\text{W}/(\text{cm}\cdot{}^\circ\text{C})$, 并且室温 $u_b=20^\circ\text{C}$.

建议习题

1. 一开始, 冷却片尺寸为 2cm × 2cm, 厚度是 1mm. 假定沿着整个左边界输入 5W 能量, 就好像这个冷却片连接到了长度 $L=2\text{cm}$ 的 CPU 芯片中的散热器一样. 在 x 方向和 y 方向取 $M=N=10$ 步, 解偏微分方程 (8.46). 用 `mesh` 命令描绘 xy 平面上的热分布. 冷却片的最高温度是几摄氏度?
2. 把冷却片的尺寸增大到 4cm × 4cm. 在冷却片的左边界沿着区间 [0, 2] 输入 5W 能量, 取相同的 $M=N=10$. 描绘热的分布, 并求出最高温度. 增加 M 和 N 的值进行实验, 解的改变有多大?
3. 在保持最高温度低于 80°C 时, 求 4cm × 4cm 冷却片能够耗散的最大能量. 假定外界温度是 20°C, 并且假定如同第 1 步和第 2 步, 能量沿着 2cm 输入.
4. 用铜片代替铝片, 导热率 $K=3.85\text{W}/(\text{cm}\cdot{}^\circ\text{C})$, 当保持最高温度低于 80°C 时, 在最佳处输入 2W 能量, 求 4cm × 4cm 冷却片能够耗散的最大能量.

   为台式计算机和便携式计算机设计冷却片的编制程序是一个吸引人的工程问题. 为了发散比以前更多的热量, 在一个小空间里需要几块冷却片, 并且在靠近冷却片边界处使用风扇以增强对流, 加入风扇使冷却片几何形状复杂化, 就把模拟引到计算流体力学的领域, 这是一个现代应用数学的极重要的领域.

8.3.2 椭圆型方程的有限元方法

解偏微分方程的有限元方法比有限差分法有着关键性的优点. 例如, 结果得到的线性方程组即使在基础几何形状复杂时也有对称结构的矩阵.

通过使用 Galerkin 方法来应用有限元素. 我们在第 7 章已对常微分方程边值问题引入了 Galerkin 方法. 尽管簿记要求更广泛, 但是对偏微分方程使用这种方法的步骤却相同. 考虑椭圆型方程

$$\Delta u+r(x,y)u=f(x,y), \tag{8.48}$$

函数 $u(x,y)$ 在平面的有界区域 R 内, R 的边界是逐段光滑的封闭曲线 S. 在边界 $S=S_1\cup S_2$ 上典型的条件是, 在 S_1 上 $u(x,y)=g_1(x,y)$(Dirichlet) 以及在 S_2 上 $\frac{\partial u}{\partial n}(x,y)=g_2(x,y)$(Neumann), 这里 n 表示外向法向量.

如在第 7 章一样, 我们将应用在区域 R 上的 L^2 函数空间. 令

$$L^2(R) = \left\{ \text{函数}\varphi(x,y)\text{在}R\text{上} \middle| \iint_R \varphi(x,y)^2 \mathrm{d}x\mathrm{d}y\text{存在且有限} \right\}.$$

我们的目标是：通过使残差 $r = \Delta u(x,y) + r(x,y)u(x,y) - f(x,y)$ 与 $L^2(R)$ 的一个大的子空间正交, 而使椭圆型方程 (8.48) 的误差的平方极小化. 令 $\varphi_1(x,y)$, $\varphi_2(x,y), \cdots, \varphi_p(x,y)$ 是 $L^2(R)$ 的一个子集, 其中每一个都在边界 S_1 上取零值. 正交化假定取为形式

$$\iint_R (\Delta u + ru - f)\varphi_i \mathrm{d}x\mathrm{d}y = 0$$

或者

$$\iint_R (\Delta u + ru)\varphi_i \mathrm{d}x\mathrm{d}y = \iint_R f\varphi_i \mathrm{d}x\mathrm{d}y, \tag{8.49}$$

其中 $1 \leqslant i \leqslant p$. 形式 (8.49) 称为椭圆型方程 (8.48) 的**弱形式**.

以下定理包括了用 Galerkin 方法需要的分步积分.

定理 8.7 (Green 第一恒等式) *设 R 是边界 S 逐段光滑的有界区域. u, v 是分段光滑函数, n 表示沿边界的单位外向法向量. 那么*

$$\iint_R v\Delta u = \int_S v\frac{\partial u}{\partial n} - \iint_R \nabla u \cdot \nabla v.$$

方向导数可以计算为

$$\frac{\partial u}{\partial n} = \nabla u \cdot (n_x, n_y),$$

这里 (n_x, n_y) 表示 R 的边界上的外向法向量. 把上述 Green 恒等式用到 (8.49) 就得到

$$\int_{S_2} \varphi_i g_2 \mathrm{d}S - \iint_R (\nabla u \cdot \nabla \varphi_i)\mathrm{d}x\mathrm{d}y + \iint_R ru\varphi_i \mathrm{d}x\mathrm{d}y = \iint_R f\varphi_i \mathrm{d}x\mathrm{d}y, \tag{8.50}$$

$i = 1, \cdots, p$, 其中用到了在 S_1 上 φ_i 等于零这一事实.

有限元方法的本质是把

$$u(x,y) = \sum_{i=1}^{q} c_i\varphi_i(x,y) \tag{8.51}$$

代入偏微分方程的弱形式, 其中 c_i 是常数, $p \leqslant q$. 回忆一下, $\varphi_i (1 \leqslant i \leqslant p)$ 在边界上等于零. 通过配置, 函数 $\varphi_i (p+1 \leqslant i \leqslant q)$ 将被用来满足在 S_1 的 Dirichlet 边界条件. 一旦确定了常数 $c_{p+1}, \cdots, c_q$, 我们将去求出其余的 $c_1, \cdots, c_p$.

把 (8.51) 代入 (8.50), 结果是

$$\int_{S_2} \varphi_i g_2 \mathrm{d}S - \iint_R \left(\sum_{j=1}^{q} c_j \nabla\varphi_j\right) \cdot \nabla\varphi_i \mathrm{d}x\mathrm{d}y + \iint_R r \left(\sum_{j=1}^{q} c_j \varphi_j\right) \varphi_i \mathrm{d}x\mathrm{d}y$$
$$= \iint_R f\varphi_i \mathrm{d}x\mathrm{d}y, \qquad i = 1, \cdots, p.$$

把常数因子 c_j 提取出来, 并解出其中前面 p 个就得到

$$\sum_{j=1}^{q} c_j \left[\iint_R \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}x\mathrm{d}y - \iint_R r\varphi_j\varphi_i \mathrm{d}x\mathrm{d}y\right] = \int_{S_2} \varphi_i g_2 \mathrm{d}S - \iint_R f\varphi_i \mathrm{d}x\mathrm{d}y$$

或者

$$\sum_{j=1}^{p} c_j \left[\iint_R \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}x\mathrm{d}y - \iint_R r\varphi_j\varphi_i \mathrm{d}x\mathrm{d}y\right] = \int_{S_2} \varphi_i g_2 \mathrm{d}S - \iint_R f\varphi_i \mathrm{d}x\mathrm{d}y$$
$$- \sum_{j=p+1}^{q} c_j \left[\iint_R \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}x\mathrm{d}y - \iint_R f\varphi_j\varphi_i \mathrm{d}x\mathrm{d}y\right], \quad i = 1, \cdots, p. \tag{8.52}$$

我们建立了含 p 个未知量 $c_1, c_2, \cdots, c_p$ 的 p 个方程的线性方程组, 其矩阵形式是 $\boldsymbol{A}\boldsymbol{c} = \boldsymbol{b}$, 其中矩阵 $\boldsymbol{A}$ 的元素和向量 $\boldsymbol{b}$ 的元素分别是

$$a_{ij} = \iint_R \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}x\mathrm{d}y - \iint_R r\varphi_j\varphi_i \mathrm{d}x\mathrm{d}y, \tag{8.53}$$

$$b_i = \int_{S_2} \varphi_i g_2 \mathrm{d}S - \iint_R f\varphi_i \mathrm{d}x\mathrm{d}y - \sum_{j=p+1}^{q} c_j \left[\iint_R \nabla\varphi_j \cdot \nabla\varphi_i \mathrm{d}x\mathrm{d}y - \iint_R r\varphi_j\varphi_i \mathrm{d}x\mathrm{d}y\right]. \tag{8.54}$$

注意到结构矩阵 $\boldsymbol{A}$ 是对称的.

最后, 我们准备选取一组显函数来作有限元素 φ_i, 并且计划进行计算. 我们按第 7 章的引导, 选取平面中三角形区域上的线性 B 样条, x 和 y 的分段线性函数. 使用区域 R 内的结点 $v_1, \cdots, v_p$ 和在边界上的结点 $v_{p+1}, \cdots, v_q$ 来形成区域 R 的三角形剖分.

具体而言, 我们将考虑矩形区域 R 上的纯 Dirichlet 问题, 即令 S_2 为空集. 考虑图 8-14 中的 $M \times N$ 网格. 令 $m = M - 1, n = N - 1$. 图 8-14a 表示矩形区域的一种可能的三角形剖分. 我们提出一种三角形的编号系统, 即从左到右开始且由下而上移动.

设 $\varphi_1, \cdots, \varphi_q$ 是分段线性函数, 满足 $\varphi_i(v_i) = 1$, 当 $j \neq i$ 时 $\varphi_i(v_j) = 0$, 并且在图 8-14a 中每个三角形上是线性的. 从而, 除了在三角形的边上外, 每一个 $\varphi_i(x, y)$ 都是可微的, 因此是 Riemann 可积函数. 根据定义, 它们满足

$$\sum_{i=1}^{q} c_i \varphi_i(v_j) = c_j, \qquad j = 1, \cdots, q.$$

根据假设, 在顶点 v_j 处的解的近似值 $u(v_j)$ 就是 c_j. 一旦解出方程组 $\boldsymbol{Ac}=\boldsymbol{b}$, 就可得到近似解.

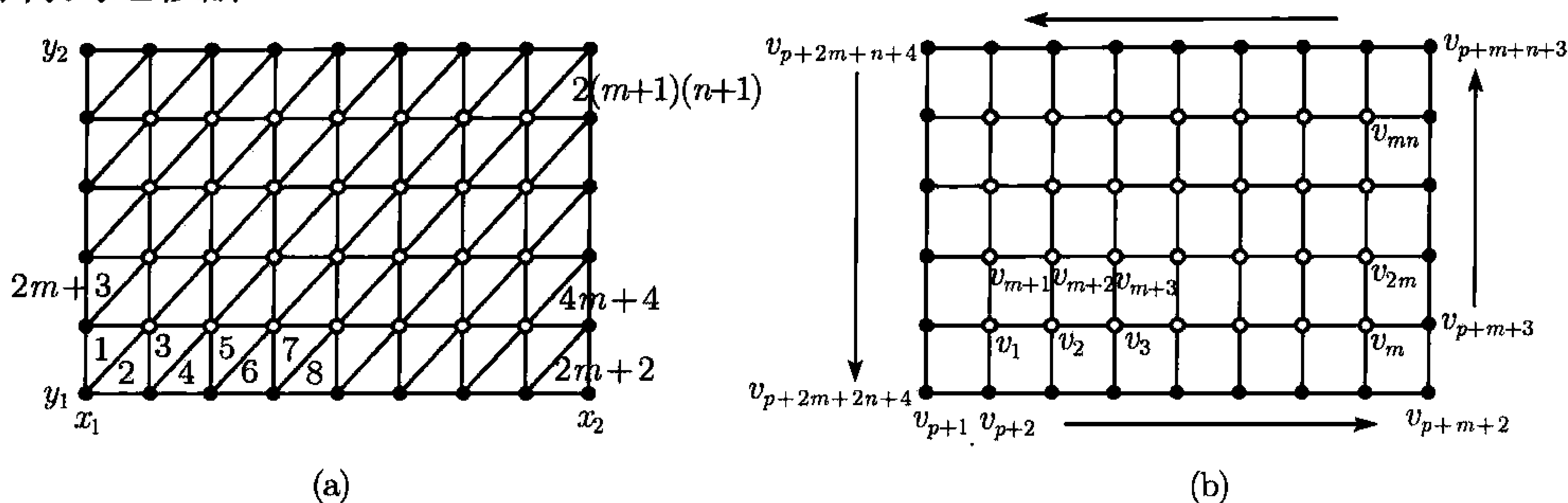


图 8-14　带 Dirichlet 边界条件的椭圆型方程的有限元解法: (a) 矩形区域三角形剖分及编号系统;(b) 顶点编号系统, 空心圆表示顶点 $v_1,\cdots,v_p$; 实心圆表示边界上的顶点 $v_{p+1},\cdots,v_q$, 其中 $q=(m+2)(n+2)$

首先可以由在 S_1 上的 Dirichlet 边界条件

$$c_j=g_1(v_j),\qquad p+1\leqslant j\leqslant q$$

确定常数 $c_{p+1},\cdots,c_q$. 然后把这些值代入 (8.52) 的右端, 去解 p 个方程 (的方程组) 的解 $c_1,\cdots,c_p$. 余下的问题是计算矩阵 (8.53) 和向量 (8.54), 以及解方程组 $\boldsymbol{Ac}=\boldsymbol{b}$.

矩阵 $\boldsymbol{R}$ 是三角形的组合. 图 8-14a 中有 $2(m+1)(n+1)$ 个三角形. 矩阵元素可以按所显示的次序, 通过在每个三角形上计算得到. 依次求出每一个积分并把其贡献加到 a_{ij} 和 b_i 中去.

用二维中点法则来近似计算分段线性函数的积分. 定义平面区域 R 的重心 (barycenter) 为点 $(\bar{x},\bar{y})$, 其中

$$\bar{x}=\frac{\displaystyle\iint_R x\mathrm{d}x\mathrm{d}y}{\displaystyle\iint_R 1\mathrm{d}x\mathrm{d}y},\qquad \bar{y}=\frac{\displaystyle\iint_R y\mathrm{d}x\mathrm{d}y}{\displaystyle\iint_R 1\mathrm{d}x\mathrm{d}y}.$$

如果 R 是顶点为 $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ 的三角形, 那么它的重心是 (见习题 6)

$$\bar{x}=\frac{x_1+x_2+x_3}{3},\qquad \bar{y}=\frac{y_1+y_2+y_3}{3}.$$

引理 8.8　线性函数 $L(x,y)$ 在平面区域 R 上的平均值是 $L(\bar{x},\bar{y})$, 即在重心的值. 换言之,

$$\iint_R L(x,y)\mathrm{d}x\mathrm{d}y=L(\bar{x},\bar{y})\cdot(R\text{的面积}.)$$

证 设 $L(x,y)=a+bx+cy$, 则

$$\begin{aligned}\iint_R L(x,y)\mathrm{d}x\mathrm{d}y &= \iint_R (a+bx+cy)\mathrm{d}x\mathrm{d}y = a\iint_R \mathrm{d}x\mathrm{d}y + b\iint_R x\mathrm{d}x\mathrm{d}y + c\iint_R y\mathrm{d}x\mathrm{d}y \\ &= (R\text{的面积})\cdot(a+b\bar{x}+c\bar{y}).\end{aligned}$$

引理 8.8 导致第 5 章的中点法则的推广, 这对计算 (8.53) 和 (8.54) 的元素的近似值很有意义. 二元函数的 Taylor 定理告诉我们:

$$\begin{aligned}f(x,y) &= f(\bar{x},\bar{y}) + \frac{\partial f}{\partial x}(\bar{x},\bar{y})(x-\bar{x}) + \frac{\partial f}{\partial y}(\bar{x},\bar{y})(y-\bar{y}) + O((x-\bar{x})^2,(y-\bar{y})^2) \\ &= L(x,y) + O\left((x-\bar{x})^2,(y-\bar{y})^2\right).\end{aligned}$$

因此,

$$\begin{aligned}\iint_R f(x,y)\mathrm{d}x\mathrm{d}y &= \iint_R L(x,y)\mathrm{d}x\mathrm{d}y + \iint_R O\left((x-\bar{x})^2,(y-\bar{y})^2\right)\mathrm{d}x\mathrm{d}y \\ &= (R\text{的面积})\cdot L(\bar{x},\bar{y}) + O(h^2) = (R\text{的面积})\cdot f(\bar{x},\bar{y}) + O(h^2),\end{aligned}$$

其中 h 是 R 的**直径**, 即 R 内两点间的最大距离, 这里我们用到了引理 8.8, 这就是二维中点法则.

二维中点法则

$$\iint_R f(x,y)\mathrm{d}x\mathrm{d}y = R\text{的面积}\cdot f(\bar{x},\bar{y}) + O(h^2), \tag{8.55}$$

其中 $(\bar{x},\bar{y})$ 是有界区域 R 的重心, 并且 $h=(R$ 的直径$)$.

中点法则表明, 为使用有 $O(h^2)$ 阶收敛性的有限元方法, 我们只要通过计算三角形重心的积分来得到 (8.53) 和 (8.54) 中的积分的近似值. 对于 B 样条函数 φ_i, 这也是特别容易的. 下面两个引理的证明见习题.

引理 8.9 *设 $\varphi(x,y)$ 是顶点为 $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ 的三角形 T 上的线性函数, 满足 $\varphi(x_1,y_1)=1,\varphi(x_2,y_2)=0,\varphi(x_3,y_3)=0$, 则 $\varphi(\bar{x},\bar{y})=\frac{1}{3}$.*

引理 8.10 *设 $\varphi_1(x,y)$ 和 $\varphi_2(x,y)$ 是顶点为 $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ 的三角形 T 上的线性函数, 满足 $\varphi_1(x_1,y_1)=1,\varphi_1(x_2,y_2)=0,\varphi_1(x_3,y_3)=0,\varphi_2(x_1,y_1)=0,\varphi_2(x_2,y_2)=1$ 以及 $\varphi_2(x_3,y_3)=0$. 令 $f(x,y)$ 是可积函数. 设*

$$d=\det\begin{bmatrix}1 & 1 & 1\\ x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\end{bmatrix}.$$

则

(a) 三角形 T 有面积 $\frac{|d|}{2}$,

(b) $\nabla\varphi_1(x,y)=\left(\dfrac{y_2-y_3}{d},\dfrac{x_3-x_2}{d}\right)$,

(c) $\displaystyle\iint_T\nabla\varphi_1\cdot\nabla\varphi_1\mathrm{d}x\mathrm{d}y=\frac{(x_2-x_3)^2+(y_2-y_3)^2}{2|d|}$,

(d) $\displaystyle\iint_T\nabla\varphi_1\cdot\nabla\varphi_2\mathrm{d}x\mathrm{d}y=\frac{-(x_1-x_3)(x_2-x_3)-(y_1-y_3)(y_2-y_3)}{2|d|}$,

(e) $\displaystyle\iint_T f\varphi_1\varphi_2\mathrm{d}x\mathrm{d}y=f(\bar{x},\bar{y})|d|/18+O(h^2)$,

(f) $\displaystyle\iint_T f\varphi_1\mathrm{d}x\mathrm{d}y=f(\bar{x},\bar{y})|d|/6+O(h^2)$,

其中 $(\bar{x},\bar{y})$ 是 T 的重心, 并且 $h=(T$ 的直径$)$.

例 8.8 取 $M=N=4$, 用有限元方法, 近似求解在 $[0,1]\times[1,2]$ 上的 Laplace 方程 $\Delta u=0$, Dirichlet 边界条件是

$$u(x,1)=\ln(x^2+1),\quad u(x,2)=\ln(x^2+4),$$
$$u(0,y)=2\ln y,\quad u(1,y)=\ln(y^2+1).$$

由于 $m=n=3$, 所以要确定 $q=(m+2)(n+2)=25$ 个系数, 其中 16 个 $(c_{10},\cdots,c_{25})$ 由配置得到, 本例中化为分别计算在 $v_{10},\cdots,v_{25}$ 处的边界条件. 对 $c_1,\cdots,c_9$ 建立矩阵方程, 从 (8.53) 和 (8.54) 简化得到

$$a_{ij}=\iint_R\nabla\varphi_i\nabla\varphi_j\mathrm{d}x\mathrm{d}y,\qquad b_i=-\sum_{j=10}^{25}c_j\iint_R\nabla\varphi_i\nabla\varphi_j\mathrm{d}x\mathrm{d}y,$$

$1\leqslant i,j\leqslant 9$. 按照引理 8.10 的计算, 矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{b}$ 的结果是在 (8.42) 中对应地加一负号. 解 $\boldsymbol{Ac}=\boldsymbol{b}$ 得到

$$\begin{aligned}
c_{13}&=1.139\,0, & c_{23}&=1.197\,4, & c_{33}&=1.287\,8,\\
c_{12}&=0.837\,6, & c_{22}&=0.915\,9, & c_{32}&=1.034\,1,\\
c_{11}&=0.484\,7, & c_{21}&=0.594\,4, & c_{31}&=0.753\,9
\end{aligned}$$

这与习题 8.6 的结果一致. ◀

下面给出矩形区域上带有 Dirichlet 边界条件 (的 Laplace 方程) 的有限元解法的 MATLAB 执行程序. 引理 8.10 提供了计算 (8.53) 和 (8.54) 中各项所需要的信息.

这个程序的前一半建立了后一半需要的簿记. 顶点 v_i 的坐标如图 8-14 中所定义. 三角形/顶点包含矩阵 nc 是 $2(m+1)(n+1)\times 3$ 矩阵, 它的第 i 行提供三个积分表示第 i 个三角形的顶点. 确定包含矩阵后, 从 Dirichet 边界条件确定 $c_{p+1},\cdots,c_q$. 然后循环走遍所有的三角形, 利用包含矩阵以及引理 8.10 把每一种贡献加到 (8.53) 及 (8.54) 中去. 一旦形成 $\boldsymbol{A}$ 和 $\boldsymbol{b}$, 通过解线性方程组确定余下的 $c_1,\cdots,c_p$. 尽管

在编码中使用 MATLAB/运算, 但是对实际应用, 它应像在第 2 章那样, 用稀疏对称解法来代替.

```
% Program 8.4 Finite element solver for elliptic equation
% Input: rectangle domain [xl,xr]x[yb,yt], covered by MxN grid
% Output: matrix w holding solution values on MxN grid
function w=ellfem(xl,xr,yb,yt,M,N)
m=M-1;n=N-1;
x=xl+(xr-xl)*(0:M)/M;                  % set mesh values
y=yb+(yt-yb)*(0:N)/N;
a=zeros(m*n,m*n);b=zeros(m*n,1); % initialize str. matrix and load
% Set (x,y) coordinates of vertices
for i=1:m
  for j=1:n
    v(m*(j-1)+i,:)=[x(i+1) y(j+1)];
  end
end
for i=1:m+2
  v(m*n+i,:)=[x(i) y(1)];
  v((m+1)*(n+1)+i+1,:)=[x(m+3-i) y(n+2)];
end
for j=1:n
  v(m*(n+1)+j+2,:)=[x(m+2) y(j+1)];
  v((m+1)*(n+2)+j+2,:)=[x(1) y(n+2-j)];
end
% Define triangle/vertex inclusion matrix nc
nc=zeros((m+2)*(n+2),3);
for i=2:m  % CORE
  for j=1:n-1
    nc(2*j*(m+1)+2*i-1,:)=[m*(j-1)+i-1 m*j+i-1 m*j+i];
    nc(2*j*(m+1)+2*i,:)=[m*(j-1)+i-1 m*(j-1)+i m*j+i];
  end
end
for i=2:m  % BOTTOM
  nc(2*i-1,:)=[i-1 i m*n+i];
  nc(2*i,:)=[i m*n+i m*n+i+1];
end
for j=2:n  % RIGHT
  nc(2*j*(m+1)-1,:)=[(j-1)*m j*m m*(n+1)+j+2];
  nc(2*j*(m+1),:)=[(j-1)*m m*(n+1)+j+1 m*(n+1)+j+2];
end
for i=2:m  % TOP
  nc(2*n*(m+1)+2*i-1,:)=[(n-1)*m+i-1 m*(n+2)+n+4-i m*(n+2)+...
   n+5-i];
  nc(2*n*(m+1)+2*i,:)=[(n-1)*m+i-1 (n-1)*m+i m*(n+2)+n+4-i];
end
for j=2:n  % LEFT
  nc(2*(j-1)*(m+1)+1,:)=[(j-1)*m+1 m*(n+2)+2*n+5-j m*(n+2)+...
   2*n+6-j];
```

```
    nc(2*(j-1)*(m+1)+2,:)=[(j-2)*m+1 (j-1)*m+1 m*(n+2)+2*n+6-j];
  end
  % CORNERS
  nc(1,:)=[1 m*n+1 m*(n+2)+2*n+4];
  nc(2,:)=[1 m*n+1 m*n+2];
  nc(2*m+1,:)=[m m*(n+1)+1 m*(n+1)+3];
  nc(2*m+2,:)=[m*(n+1)+1 m*(n+1)+2 m*(n+1)+3];
  nc(2*n*(m+1)+1,:)=[m*(n+2)+n+3 m*(n+2)+n+4 m*(n+2)+n+5];
  nc(2*n*(m+1)+2,:)=[m*(n-1)+1 m*(n+2)+n+3 m*(n+2)+n+5];
  nc(2*(n+1)*(m+1)-1,:)=[m*n m*(n+1)+n+3 m*(n+1)+ n+4];
  nc(2*(n+1)*(m+1),:)=[m*n m*(n+1)+n+2 m*(n+1)+n+3];
  for i=1:m+2                           % use Dirichlet conditions
    c(m*n+i)=gbottom(v(m*n+i,1));
    c((m+1)*(n+1)+1+i)=gtop(v((m+1)*(n+1)+1+i,1));
  end
  for j=1:n
    c(m*(n+1)+2+j)=gright(v(m*(n+1)+2+j,2));
    c((m+1)*(n+2)+2+j)=gleft(v((m+1)*(n+2)+2+j,2));
  end
  mn=m*n;
  % Cycle through triangles
  for t=1:2*(m+1)*(n+1)
    d=abs(det([1 1 1;v(nc(t,1),:)' v(nc(t,2),:)' v(nc(t,3),:)']));
    bary=(v(nc(t,1),:)+v(nc(t,2),:)+v(nc(t,3),:))/3;
    for i=1:3
      j=mod(i,3)+1;
      k=mod(i+1,3)+1;
      if(nc(t,i)<=mn)
        a(nc(t,i),nc(t,i))=a(nc(t,i),nc(t,i))...
          +((v(nc(t,j),2)-v(nc(t,k),2))^2 ...
          +(v(nc(t,j),1)-v(nc(t,k),1))^2)/(2*d)-r(bary)*d/18;
        b(nc(t,i))=b(nc(t,i))-f(bary)*d/6;
      end
    end
    for i=1:3
      j=mod(i,3)+1;
      k=mod(i+1,3)+1;
      if(nc(t,i)<=mn & nc(t,j)<=mn)
        a(nc(t,i),nc(t,j))=a(nc(t,i),nc(t,j))...
          -((v(nc(t,i),1)-v(nc(t,k),1))...
          *(v(nc(t,j),1)-v(nc(t,k),1))...
          +(v(nc(t,i),2)-v(nc(t,k),2))...
          *(v(nc(t,j),2)-v(nc(t,k),2)))/(2*d)-r(bary)*d/18;
        a(nc(t,j),nc(t,i))=a(nc(t,i),nc(t,j));
      end
      if(nc(t,i)<=mn & nc(t,j)>mn)
        b(nc(t,i))=b(nc(t,i))+c(nc(t,j))...
          *((v(nc(t,i),1)-v(nc(t,k),1))...
          *(v(nc(t,j),1)-v(nc(t,k),1))...
```

```
        +(v(nc(t,i),2)-v(nc(t,k),2))...
        *(v(nc(t,j),2)-v(nc(t,k),2)))/(2*d)+r(bary)*d/18;
    end
        if(nc(t,i)<=mn & nc(t,k)>mn)
          b(nc(t,i))=b(nc(t,i))+c(nc(t,k))...
            *((v(nc(t,i),1)-v(nc(t,j),1))...
            *(v(nc(t,k),1)-v(nc(t,j),1))...
            +(v(nc(t,i),2)-v(nc(t,j),2))...
            *(v(nc(t,k),2)-v(nc(t,j),2)))/(2*d)+r(bary)*d/18;
        end
      end
    end
    u=a\b;
    for i=1:m
        for j=1:n
           w1(i+(j-1)*m)=log(x(i+1)^2+y(j+1)^2);
           w(i,j)=u(i+(j-1)*m);
        end
    end
    w1=[gbottom(x(2:M))' w gtop(x(2:M))']; % plot
    w1=[gleft(y);w1;gright(y)];
    mesh(x,y,w1')

    function u=f(x)
    u=0;

    function u=r(x)
    u=0;

    function u=gbottom(x)
    %Use dot notation
    u=log(x.^2+1);

    function u=gtop(x)
    u=log(x.^2+4);

    function u=gleft(y)
    u=2*log(y);

    function u=gright(y)
    u=log(y.^2+1);
```

偏微分方程用于模拟工程和科学中的时间和空间现象. 模拟不可压缩流体的 Navier-Stokes 方程用于研究薄膜涂层、润滑、动脉中血液的原动力以及到天体气体的湍流等主题. 改进有限差分和有限元解法仍然是计算研究领域最活跃的领域.

习题 8.3

1. 证明 $u(x,y)=\ln(x^2+y^2)$ 是满足例 8.6 中 Dirichlet 边界条件的 Laplace 方程的解.

2. 证明 (a) $u(x,y)=x^2y-\frac{1}{3}y^3$ 及 (b) $u(x,y)=\frac{1}{6}x^4-x^2y^2+\frac{1}{6}y^4$ 是调和函数.

3. 证明函数 (a) $u(x,y)=\mathrm{e}^{-\pi y}\sin\pi x$, (b) $u(x,y)=\sinh\pi x\sin\pi y$ 分别是带有以下给定边界条件的 Laplace 方程的解:

$$\text{(a)}\begin{cases} u(x,0)=\sin\pi x, & 0\leqslant x\leqslant 1,\\ u(x,1)=\mathrm{e}^{-\pi}\sin\pi x, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ u(1,y)=0, & 0\leqslant y\leqslant 1;\end{cases}\qquad \text{(b)}\begin{cases} u(x,0)=0, & 0\leqslant x\leqslant 1,\\ u(x,1)=0, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ u(1,y)=\sinh\pi\sin\pi y, & 0\leqslant y\leqslant 1.\end{cases}$$

4. 证明函数 (a) $u(x,y)=\mathrm{e}^{-xy}$, (b) $u(x,y)=(x^2+y^2)^{\frac{3}{2}}$ 分别是带有以下给定边界条件的 Poisson 方程的解

$$\text{(a)}\begin{cases} \Delta u=\mathrm{e}^{-xy}(x^2+y^2), & \\ u(x,0)=1, & 0\leqslant x\leqslant 1,\\ u(x,1)=\mathrm{e}^{-x}, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ u(1,y)=\mathrm{e}^{-y}, & 0\leqslant y\leqslant 1;\end{cases}\qquad \text{(b)}\begin{cases} \Delta u=9\sqrt{x^2+y^2}, & \\ u(x,0)=x^3, & 0\leqslant x\leqslant 1,\\ u(x,1)=(1+x^2)^{\frac{3}{2}}, & 0\leqslant x\leqslant 1,\\ u(0,y)=y^3, & 0\leqslant y\leqslant 1,\\ u(1,y)=(1+y^2)^{\frac{3}{2}}, & 0\leqslant y\leqslant 1.\end{cases}$$

5. 证明函数 (a) $u(x,y)=\sin\frac{\pi}{2}xy$, (b) $u(x,y)=\mathrm{e}^{xy}$ 分别是带有以下给定 Dirichlet 边界条件的椭圆型方程的解:

$$\text{(a)}\begin{cases} \Delta u+\frac{\pi^2}{4}(x^2+y^2)u=0, & \\ u(x,0)=0, & 0\leqslant x\leqslant 1,\\ u(x,1)=\sin\frac{\pi}{2}x, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ u(1,y)=\sin\frac{\pi}{2}y, & 0\leqslant y\leqslant 1;\end{cases}\qquad \text{(b)}\begin{cases} \Delta u=(x^2+y^2)u, & \\ u(x,0)=1, & 0\leqslant x\leqslant 1,\\ u(x,1)=\mathrm{e}^{x}, & 0\leqslant x\leqslant 1,\\ u(0,y)=1, & 0\leqslant y\leqslant 1,\\ u(1,y)=\mathrm{e}^{y}, & 0\leqslant y\leqslant 1.\end{cases}$$

6. 证明顶点为 $(x_1,y_1),(x_2,y_2),(x_3,y_3)$ 的三角形的重心是 $\bar{x}=(x_1+x_2+x_3)/3,\bar{y}=(y_1+y_2+y_3)/3$.

7. 证明引理 8.9.

8. 证明引理 8.10.

计算机问题 8.3

1. 取 $h=k=0.1$, 在 $0\leqslant x\leqslant 1, 0\leqslant y\leqslant 1$ 上, 用有限差分法解习题 3 中的 Lapace 方程问题. 用 MATLAB 的 `mesh` 指令把解绘制出来.

2. 在 $0\leqslant x\leqslant 1, 0\leqslant y\leqslant 1$ 上, 用有限差分法解习题 4 中的 Poisson 方程问题, 取 $h=k=0.1$. 把解绘制出来.

3. 取 $h=k=0.1$, 用有限差分法, 从带有以下给定边界条件的 Laplace 方程中, 求出在正方形 $0\leqslant x,y\leqslant 1$ 上的静电势, 并把解绘制出来.

$$\text{(a)}\begin{cases} u(x,0)=0, & 0\leqslant x\leqslant 1,\\ u(x,1)=\sin\pi x, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ u(1,y)=0, & 0\leqslant y\leqslant 1,\end{cases}\qquad \text{(b)}\begin{cases} u(x,0)=\sin\frac{\pi}{2}x, & 0\leqslant x\leqslant 1,\\ u(x,1)=\cos\frac{\pi}{2}x, & 0\leqslant x\leqslant 1,\\ u(0,y)=\sin\frac{\pi}{2}y, & 0\leqslant y\leqslant 1,\\ u(1,y)=\cos\frac{\pi}{2}y, & 0\leqslant y\leqslant 1.\end{cases}$$

4. 取 $h=k=0.1$, 用有限差分法, 从带有以下给定边界条件的 Laplace 方程中, 求出在正方形 $0\leqslant x,y\leqslant 1$ 上的静电势, 把解绘制出来.

$$
\text{(a)}\begin{cases} u(x,0)=0, & 0\leqslant x\leqslant 1,\\ u(x,1)=x^3, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ u(1,y)=y^2, & 0\leqslant y\leqslant 1,\end{cases}\qquad \text{(b)}\begin{cases} u(x,0)=0, & 0\leqslant x\leqslant 1,\\ u(x,1)=x\sin\frac{\pi}{2}x, & 0\leqslant x\leqslant 1,\\ u(0,y)=0, & 0\leqslant y\leqslant 1,\\ y(1,y)=y, & 0\leqslant y\leqslant 1.\end{cases}
$$

5. 静水压力 (hydrostatic pressure) 可以用液压压头来表示, 它定义为相同高度 u 的一圆柱体水施加的压力. 在地下水库内, 稳定状态的地面水流满足 Laplace 方程 $\Delta u=0$. 假定水库的尺寸是 2km × 1km, 地下水位高

$$
\begin{cases} u(x,0)=0.01, & 0\leqslant x\leqslant 2,\\ u(x,1)=0.01+0.003x, & 0\leqslant x\leqslant 2,\\ u(0,y)=0.01, & 0\leqslant y\leqslant 1,\\ u(1,y)=0.01+0.006y^2, & 0\leqslant y\leqslant 1\end{cases}
$$

在水库的边界上成立, 单位是千米. 计算在水库中心的 $u\left(1,\frac{1}{2}\right)$.

6. 在一个受热的金属铜板上的稳态的温度 u 满足 Poisson 方程

$$
\Delta u=-\frac{D(x,y)}{K},
$$

这里 $D(x,y)$ 是在 (x,y) 处的能量密度, K 是热传导率. 假定板的形状是 $[0,4]\text{cm}\times[0,2]\text{cm}$ 的矩形, 它的边界保持在常温 30°C, 而且以常速 $D(x,y)=5\text{W/cm}^3$ 产生能量. 铜的热传导率 $K=3.85\text{W/(cm}\cdot{}^\circ\text{C)}$. (a) 绘出铜板上的温度分布. (b) 求在中心点 $(x,y)=(2,1)$ 处的温度.

7. 当步长 $h=k=2^{-p}(p=2,\cdots,5)$ 时, 对习题 3 中的 Laplace 方程, 作一个在 $(x,y)=\left(\frac{1}{4},\frac{3}{4}\right)$ 处的有限差分近似解及误差 (作为步长 $h=k=2^{-p}$ 的函数) 的表.

8. 当步长 $h=k=2^{-p}(p=2,\cdots,5)$ 时, 对习题 4 的 Poisson 方程, 作一个在 $(x,y)=\left(\frac{1}{4},\frac{3}{4}\right)$ 处的有限差分近似解及误差 (作为步长 $h=k=2^{-p}$ 的函数) 的表.

9. 取 $h=k=0.1$, 在 $0\leqslant x\leqslant 1, 0\leqslant y\leqslant 1$ 上, 用有限元方法解习题 3 中的 Laplace 方程问题. 用 MATLABS 的 `mesh` 命令绘出解.

10. 取 $h=k=0.1$, 在 $0\leqslant x\leqslant 1, 0\leqslant y\leqslant 1$ 上, 用有限元方法解习题 4 中的 Poisson 方程问题, 并绘出解.

11. 取 $h=k=0.1$, 用有限元方法解习题 5 中的椭圆型偏微分方程. 把解绘制出来.

12. 取 $h=k=0.1$, 用有限元方法解带有 Dirichlet 边界条件的椭圆型偏微分方程. 绘出解.

$$
\text{(a)}\begin{cases} \Delta u+\sin\pi xy=(x^2+y^2)u, \\ u(x,0)=0, \quad 0\leqslant x\leqslant 1,\\ u(x,1)=0, \quad 0\leqslant x\leqslant 1,\\ u(0,y)=0, \quad 0\leqslant y\leqslant 1,\\ u(1,y)=0, \quad 0\leqslant y\leqslant 1,\end{cases}\qquad \text{(b)}\begin{cases} \Delta u+(\sin\pi xy)u=\mathrm{e}^{2xy}, \\ u(x,0)=0, \quad 0\leqslant x\leqslant 1,\\ u(x,1)=0, \quad 0\leqslant x\leqslant 1,\\ u(0,y)=0, \quad 0\leqslant y\leqslant 1,\\ u(1,y)=0, \quad 0\leqslant y\leqslant 1.\end{cases}
$$

13. 当步长 $h=k=2^{-p}(p=2,\cdots,5)$ 时, 对习题 5 中的椭圆型方程, 作一个在 $(x,y)=\left(\frac{1}{4},\frac{3}{4}\right)$ 处的有限元方法近似解和误差 (作为步长的函数) 的表.

软件和进一步阅读

关于偏微分方程及其在科学与工程中的应用, 有丰富的文献资料. 新近的包含应用观点的教科书有 [9, 12, 7, 18, 8]. 许多教科书如 [17, 11, 10, 14] 给出了求解偏微分方程数值方法更多的内容, 比如有限差分和有限元方法. [3, 1, 16] 主要关注有限元方法.

MATLAB 中偏微分方程的工具箱非常值得推荐, 它已成为偏微分方程和工程数学课程中广受欢迎的帮手.Maple 有一个类似的软件包叫 PDEtools. 关于数值偏微分方程, 无论是一般的使用抑或是特殊问题的求解, 已经开发展出了一些独一无二的软件包. 求解平面中一般区域上的椭圆型偏微分方程的软件包 ELLPACK[15] 和 PLTMG[2] 可供免费使用. 它们在 Netlib 都可得到.

有限元方法软件包括免费软件 FEAST (Finite Element and Solution Tools, 有限元和解工具), Free FEM 和 PETSc (Portable Extensible Toolkit for Scientific Computing, 求解科学计算问题的便携可扩展的工具箱), 以及商用软件 FEMALAB、NASTRAN 和 DIFFPACK, 等等.IMSL 程序库包含求解矩形区域上的 Poisson 方程的程序 DFPS2H, 及三维空间中 Poisson 方程的 DFPS3H. 这些方法都基于有限差分.

NAG 程序库包含有有限差分和有限元的一些程序. 程序 D03EAF 利用一个积分方程方法求解二维的 Laplace 方程; D03EEF 利用一个 7 点有限差分公式, 处理多类边值条件. 程序 D03PCF 与 D03PFF 分别处理抛物型与双曲型方程.

第 9 章　随机数及其应用

布朗运动是描述不确定性行为的模型, 由 Robert Brown 在 1827 年提出. 他的最初目的是为了研究水面上花粉微粒和附近分子碰撞的不规则运动. 而该模型的应用已经远远超过了其最初的研究背景.

如今金融分析师们用同样的方法研究资产的定价, 把其看作是由于众多投资者不一致的投资活动所导致的易变实体. 1973 年, Fischer Black 和 Myron Scholes 最先利用指数布朗运动提供股票期权的精确定价. 作为一项重要创新, Black-Scholes 公式迅速被编入到第一批便携式计算器, 并用于华尔街的交易大厅. 该项成果在 1997 年被授予诺贝尔经济学奖, 并已经深入到了金融理论和实践.

实例检验　9.4.2 节后的实例检验 9研究了蒙特卡罗模拟和这个著名的公式.

前 3 章中考虑了由微分方程决定的确定性模型. 给定适当的初值和边界条件, 从数学上讲其解是确定的, 并且对于给定的精度, 可以由合适的数值方法求解. 另一方面, 一个随机模型又包含了不确定性, 因为它把噪声引入了其定义.

随机系统的计算机模拟需要通过生成随机数来模拟噪声. 本章以一些关于随机数的基本事实以及它们在模拟中的使用作为开始. 9.2 节包括了随机数的一个重要应用蒙特卡罗模拟. 9.3 节介绍了随机游动和布朗运动. 9.4 节涵盖了随机微积分学的基本思想, 以及许多随机微分方程 (SDE) 的标准例子, 这些例子在物理、生物和金融学上已被证明是有用的. SDE 解法是以第 6 章中 ODE 的解法为基础, 但引入了噪声部分.

概率论的一些基本概念会在本章中用到. 这些基本概念, 例如期望、方差和随机变量的独立性等, 在 9.2 节至 9.4 节中是重要的.

9.1　随　机　数

人们对随机数的含义都有一个直观的认识, 但是要给它下一个精确的定义却非常困难. 要找到一种简单有效的方法来生成随机数也并不容易. 即使给定规则, 计算机程序也无法生成真正的随机数, 其他方法同样如此. 下面将解决如何生成这样的伪随机数, 即通过简单的方法, 用确定的计算机程序来生成数列, 使其看起来尽可能像是随机的.

随机数生成方法的目标是为了使输出的数字独立同分布. “独立” 是指每个新

生成的数 x_n 不依赖于前一个数字 x_{n-1}，事实上不依赖于前面的所有数字 x_{n-1}, $x_{n-2}, \cdots$. “同分布” 是指若重复生成 x_n 多次, 其分布直方图跟 x_{n-1} 的分布直方图应该是非常相似的. 换句话说, 独立性是指 x_n 独立于 x_{n-1}, x_{n-2} 等, 同分布是指 x_n 的分布独立于 n. 所得到的直方图或者分布可能是 0 到 1 之间实数的均匀分布, 或更复杂些, 例如可能是正态分布.

当然, 随机数定义中的独立性部分与实际中基于计算机生成随机数的方法是不一致的, 后者生成的是完全可预测、可重复的数据流. 事实上, 对某些模拟来说, 可重复性是非常有用的. 技巧就是使数字*看起来*相互独立, 尽管生成的方法一点也不独立. **伪随机数** (pseudo-random) 就是为这样的情况而提出的——在独立同分布的意义上, 使确定的生成数尽量具有随机性.

想要利用高度相关的办法来生成不相关的随机数的事实, 也解释了为什么不存在完美的、基于软件的并且通用的生成随机数的方法. 正如 John Von Neumann 在 1951 年所说的, “用算术方法生成随机数的人, 从某种程度上讲就是一种过失.” 这里希望达到的目标是, 用户通过使用随机数想要检验的特殊假设对相关性和生成方法不敏感.

随机数是从确定的概率分布选取的代表. 对于分布的选取, 有多种可能的选择. 这里我们限制在两种概率分布：均匀分布和正态分布.

9.1.1　伪随机数

最简单的随机数的集合是区间 [0, 1] 上的均匀分布. 这些数字就好像是放在一个看不见的区间上, 所以从中选取每个数字都是等可能的. 如何用计算机程序生成这样的数列呢？

这里有一种方法来生成 [0, 1] 上的均匀 (伪) 随机数. 选取一个初始的整数 $x_0 \neq 0$, 称为**种子** (seed). 然后通过下面的迭代生成数列 u_i：

$$\begin{cases} x_i = 13x_{i-1} \pmod{31}, \\ u_i = \dfrac{x_i}{13}, \end{cases} \tag{9.1}$$

即用 13 乘以 x_{i-1}, 然后做模运算, 把它除以 31 取余得到下一个随机数. 得到的序列在 30 个非零的数字 $1/30, \cdots, 30/31$ 之间重复取值. 或者说, 该随机数生成方法的**周期** (period) 是 30. 对于该数列, 没有什么看起来比其更具有随机性. 一旦种子选定, 那么 30 个可能数字的循环顺序也就确定了. 最早的随机数生成方法遵循同样的逻辑, 尽管其具有更大的周期.

以 $x_0 = 3$ 作为种子, 表 9-1 是由该方法生成的前 10 个数字.
以 $3 * 13 = 39 \to 8 \pmod{31}$ 为开始, 产生随机数 $8/31 \approx 0.258\,1$. 第二个随机数 $8 * 13 = 104 \to 11 \pmod{31}$, 生成 $11/31 \approx 0.354\,8$. 依次下去, 数列便在 30 个可能

的数之间取值.

表 9-1

x	u	x	u
8	0.258 1	17	0.548 4
11	0.354 8	4	0.129 0
19	0.612 9	21	0.677 4
30	0.967 7	25	0.806 5
18	0.580 6	15	0.483 9

该例子是随机数生成方法的一个最基本的类型.

定义 9.1 线性同余生成元 (linear congruential generator, LCG) 具有形式

$$\begin{cases} x_i = ax_{i-1} + b \pmod{m} \\ u_i = \dfrac{x_i}{m}, \end{cases} \tag{9.2}$$

其中 a 为**乘子** (multiplier), b 是**偏移量** (offset), m 为**模数** (modulus).

在前面的生成元中, $a = 13$, $b = 0$, $m = 31$. 在下面两个例子中仍保持 $b = 0$. 常见的做法是把一个不大的非零数 b 加到随机数生成序列中去.

随机数的一个应用是用定义域中的随机数来逼近函数的平均值. 这是蒙特卡罗技术最简单的形式, 9.2 节会更详细地讨论它.

例 9.1 在 [0, 1] 上, 近似计算在函数 $y = x^2$ 曲线下方的区域面积.

由定义, 一个函数在 $[a, b]$ 上的平均值为

$$\frac{1}{b-a}\int_a^b f(x)\mathrm{d}x$$

所以问题中的所求面积就是 $y = x^2$ 在 [0, 1] 上的平均值. 该平均值可以由区间上随机点处的函数值的平均值来逼近, 如图 9-1 所示. 对于前 10 个均匀随机数, 函数平均值

$$\frac{1}{10}\sum_{i=1}^{10} f(u_i)$$

为 0. 350, 与准确值 1/3 差不多. 在该结果中使用所有的 30 个随机数的平均值, 可得到改进后的估计为 0.328. ◀

称例 9.1 中的应用为蒙特卡罗类型 1 问题, 其将问题简化为求函数的平均值. 注意到我们已经使用了生成元 (9.1) 可以提供的所有 30 个随机数, 所以要提高精度, 就需要更多的随机数. 可以继续用 LCG 模型, 但乘子 a 和模数 m 需要增加.

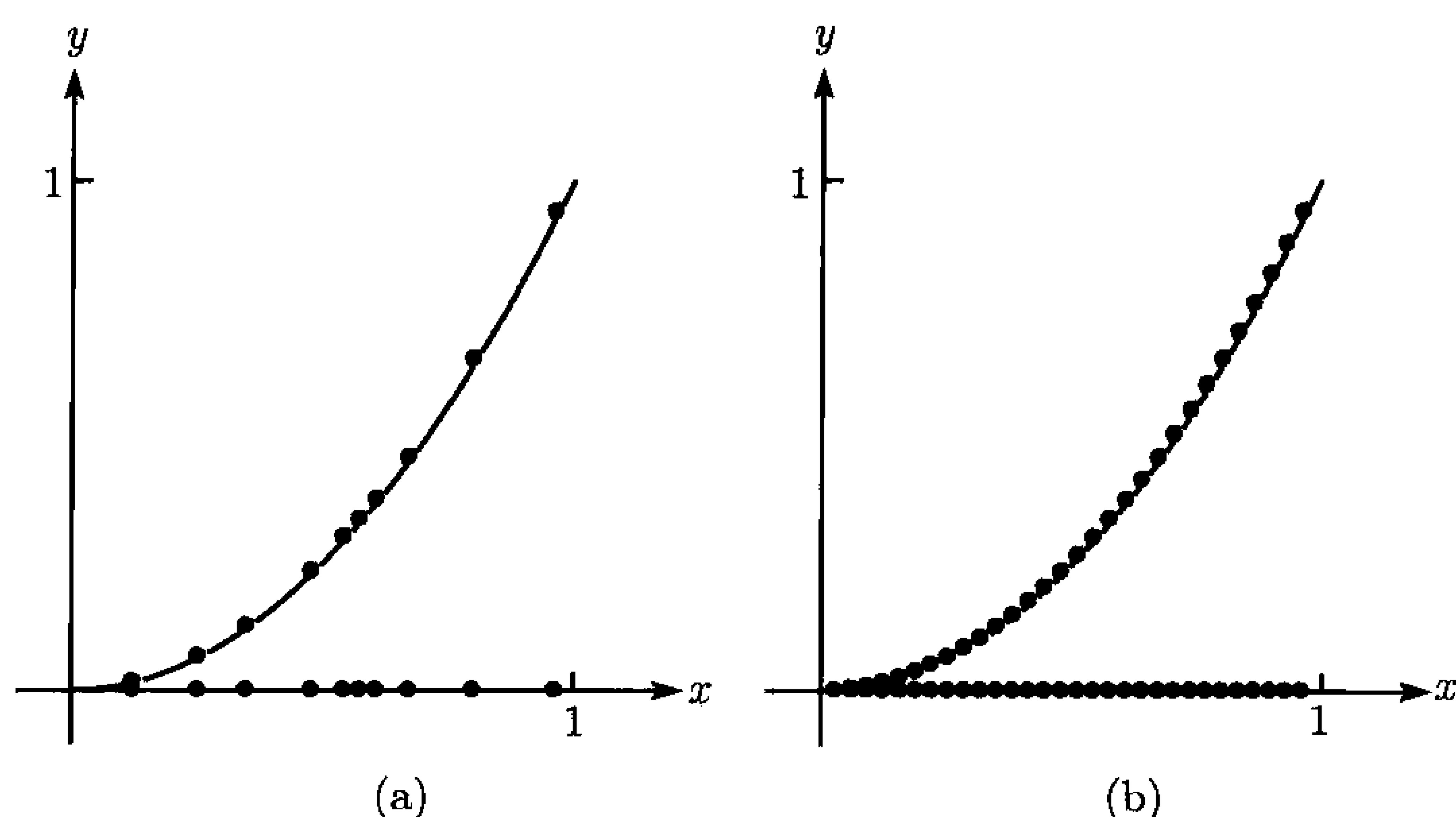


图 9-1 用随机数对函数作平均：(a) 基本生成元 (9.1) 中前 10 个随机数的函数平均值为 0.350, 其中种子为 $x_0 = 3$; (b) 采用所有 30 个数, 得到更准确的结果为 0.328

Pank 和 Miller[18] 提出了一个线性同余生成元, 通常称为 "最小标准" 生成元, 因为它用非常少的代码得到尽可能好的结果. 该随机数生成元用在了 20 世纪 90 年代的 MATLAB版本 4 中.

最小标准随机数生成元

$$\begin{cases} x_i = ax_{i-1} + b \pmod{m} \\ u_i = \dfrac{x_i}{m}, \end{cases} \tag{9.3}$$

其中 $m = 2^{31} - 1, a = 7^5 = 16\,807, b = 0$.

如果一个素数具有 $2^p - 1$ 的形式, 其中 p 是整数, 就称之为**梅森素数** (Mersenne prime). 欧拉在 1772 年发现了梅森素数. 最小标准随机数生成元的重复时间是最大数 $2^{31} - 2$, 即当种子非零时, 当它再次取到该数时已经取遍了最大数以下的所有整数. 这大约有 2×10^9 个数字, 对 20 世纪来说可能已经足够了, 但现在一般是不够的, 因为计算机可以在每秒执行更多周期.

例 9.2 求满足下面不等式的点 (x, y) 的集合的面积.

$$4(2x-1)^4 + 8(2y-1)^8 < 1 + 2(2y-1)^3(3x-2)^2.$$

我们称之为蒙特卡罗类型 2 问题. 没有明确的方法可以通过求解一元函数值的平均值来求该区域的面积, 因为不能解出 y. 可是对给定的 (x, y), 容易判断其是否属于该集合. 可以通过判断给定的随机数对 $(x, y) = (u_i, u_{i+1})$ 是否属于该集合从而以一定的概率估计目标区域的面积.

图 9-2 显示用最小标准 LCG 生成的 10 000 个随机数对得到的区域. 在单位区

域 $0 \leqslant x, y \leqslant 1$ 中满足不等式的数对的比例, 如图所示, 约为 0.547, 它是该区域面积的一个近似解. ◀

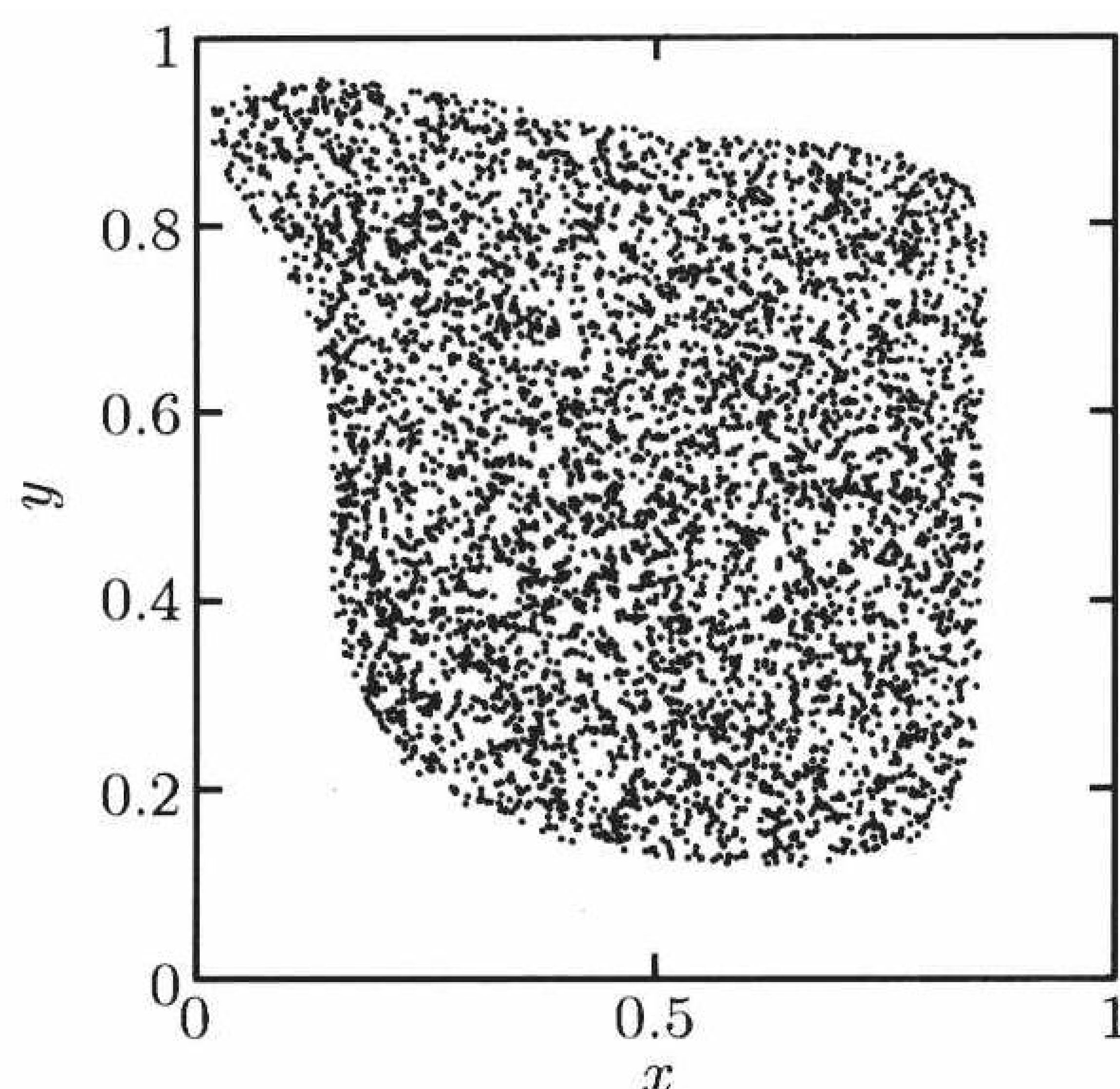


图 9-2 求面积的蒙特卡罗方法. 从 $[0,1]\times[0,1]$ 区间中选取的 10 000 个随机数对, 画出的点满足例 9.2 中的不等式. 画出的与没有画出的随机数对的比例是所求面积的近似解

尽管我们对两种类型的蒙特卡罗问题做了区别, 但是在两者之间并无确定的界限. 通常它们都用于求函数的平均值. 在前面"类型 1"的例子中这是很明显的. 在"类型 2"的例子中, 我们尝试计算所求集合的**特征函数** (characteristic function) 的平均值, 该函数在所求集合中的取值为 1, 集合外取值为 0. 这里主要的区别是, 不像例 9.1 中的函数 $f(x)=x^2$, 特征函数是不连续的, 它在集合边界处函数值有跳跃. 我们也可以很容易想象类型 1 和类型 2 的结合 (见计算机问题 9.1.8).

一个非常差的随机数生成元是 randu 生成元, 在许多早期的 IBM 电脑上使用并移植到了许多其他电脑上. 在因特网上很容易搜索到该生成元, 显然它还在使用中.

randu 生成元

$$\begin{cases} x_i = a x_{i-1} \pmod m \\ u_i = \dfrac{x_i}{m}, \end{cases} \tag{9.4}$$

其中 $a = 65\,539 = 2^{16}+3, m = 2^{31}$.

随机种子 $x_0 \neq 0$ 是任意选取的. 非素数模的选取是为了使求模运算尽可能快, 乘子的选取主要是为了数对的选取简单. 该生成元的主要问题在于它不服从随机数的独立性假定. 注意到

$$a^2-6a=(2^{16}+3)^2-6(2^{16}+3)=2^{32}+6\times2^{16}+9-6\times2^{16}-18=2^{32}-9,$$

所以 $a^2-6a+9=0 \pmod m$, 因此

$$x_{i+2}-6x_{i+1}+9x_i = a^2x_i - 6ax_i + 9x_i \pmod m = 0 \pmod m.$$

除以 m 得

$$u_{i+2} = 6u_{i+1} - 9u_i \pmod 1. \tag{9.5}$$

问题不在于 u_{i+2} 可以由前面生成的两个数字决定. 当然它甚至可由前一个数字决定, 因为生成元是确定的. 问题在于关系式 (9.5) 中的系数较小, 它导致了随机数之间的相关性变得显而易见. 图 9-3a 给出了由 randu 生成元的 10 000 个随机数, 并以三元组 (u_i, u_{i+1}, u_{i+2}) 的形式绘出. 关系式 (9.5) 的一个结果是随机数的所有三元组都将分布在 15 个平面中的一个上, 如图所示. 事实上, 因为 $u_{i+2} - 6u_{i+1} + 9u_i$ 必须是整数, 仅有的可能取值必须在 -5 (当 u_{i+1} 相对大, 而 u_i, u_{i+2} 较小时) 和 $+9$ (相反情形) 之间. 平面 $u_{i+2} - 6u_{i+1} + 9u_i = k$, 其中 $-5 \leqslant k \leqslant 9$, 便是图 9-3 中的 15 个平面. 习题 9.1.5 要求分析另一个著名的具有类似缺陷的随机数生成元.

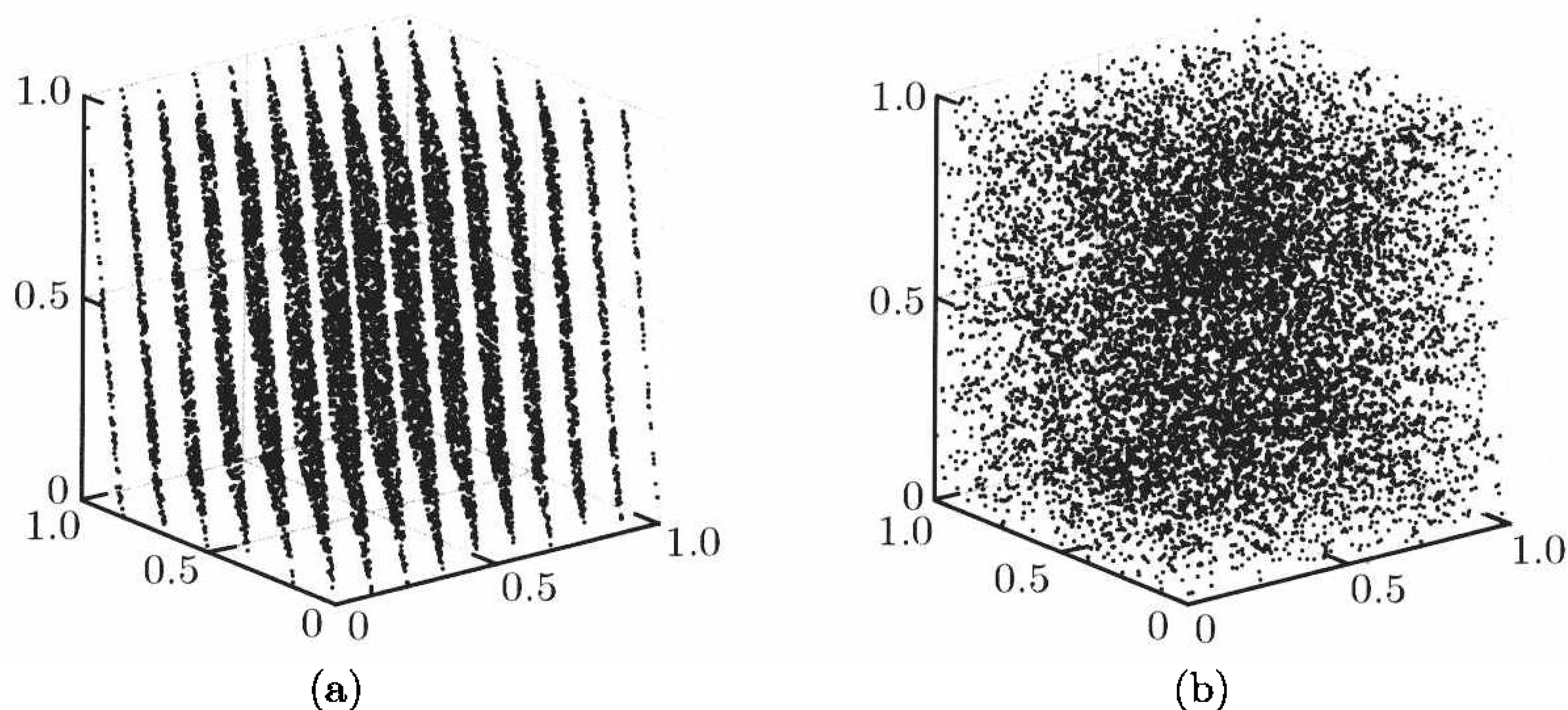


图 9-3　两种随机数生成元的比较: 图 (a) 和 (b) 分别描绘出了randu 生成元和最小标准生成元的 10 000 个三元数组 (u_i, u_{i+1}, u_{i+2})

最小标准 LCG 不存在这样的问题, 至少不存在严重程度与此相同的问题. 因为 (9.3) 中的 m 和 a 是互素的, 像 (9.5) 中那样带有小系数的相邻 u_i 的关系很难确定, 相邻的 3 个随机数之间的关系也就更复杂. 图 9-3b 把由最小标准生成元得到的 10 000 个随机数与由 randu 生成元得到的随机数进行了比较.

例 9.3　用 randu 生成元近似求解圆心为 (1/3, 1/3, 1/2)、半径为 0. 04 的球的体积.

尽管该球的体积非零, 但直接用 randu 生成元得到的结果却是零. 蒙特卡罗逼近是在三维单位立方体中生成随机点, 并记录生成的点落入球中的比例作为近似的体积.

点 (1/3, 1/3, 1/2) 在平面 $9x - 6y + z = 1$ 和 $9x - 6y + z = 2$ 之间, 到每个平面的距离为 $1/\left(2\sqrt{118}\right) \approx 0.046$. 所以由 randu 生成元产生的三维点 $(x, y, z) = (u_i, u_{i+1}, u_{i+2})$ 不能落入给定的球里. 由于随机数生成元的选取, 该问题的蒙特卡罗逼近非常不成功. 奇怪的是在 20 世纪六七十年代期间, 当这种生成元严重依赖

于计算机模拟时, 这类困难竟然大多没有被注意到. ◀

在 MATLAB 现在的版本中, 随机数不再由 LCG 产生. 从 MATLAB 5 开始, 由 G.Marsaglia 和其他人 [14] 发展的延迟 Fibonacci 生成元 (lagged Fibonacci generator) 用在了 `rand` 命令中. 其用到了 0 到 1 之间所有可能的浮点数. MATLAB 声称该方法的周期大于 2^{1400}, 这要比 MATLAB 研发以来所有的程序的总步数还要多.

到现在为止, 我们主要研究了在区间 [0, 1] 上生成伪随机数的问题. 为生成一般区间 $[a, b]$ 上的随机数的均匀分布, 只需要把区间长度拉伸到 $b-a$, 即新区间的长度. 所以, 区间 [0,1] 中生成的每一个随机数 r 可由 $(b-a)r+a$ 代替.

该方法可独立地推广到任意维数. 例如, 要生成 xy 平面上矩形 $[1,3]\times[2,8]$ 中的均匀随机数, 可以先生成均匀随机数对 r_1, r_2, 然后用 $(2r_1+1, 6r_2+2)$ 表示随机点.

9.1.2 指数随机数和正态随机数

指数随机变量 V 通过**概率分布函数** $p(x)=a\mathrm{e}^{-ax}(a>0)$ 选取正数. 换言之, 指数随机数 $r_1, r_2, \cdots, r_n$ 的分布直方图当 $n\to\infty$ 时趋于 $p(x)$.

9.1.1 节中均匀随机数的生成元, 容易生成**指数随机数**. **累积分布函数**为

$$P(x)=\mathrm{Prob}\,(V\leqslant x)=\int_0^x p(x)\mathrm{d}x=1-\mathrm{e}^{-ax}.$$

主要思想是选取指数随机变量, 以使得 $\mathrm{Prob}(V\leqslant x)$ 是 0 到 1 之间的均匀分布, 即对给定的均匀随机数 u, 设

$$u=\mathrm{Prob}(V\leqslant x)=1-\mathrm{e}^{-ax},$$

对 x 求解, 得

$$x=\frac{-\ln(1-u)}{a}. \tag{9.6}$$

因此式 (9.6) 便通过均匀随机数 u 生成了指数随机数.

该思想是普遍成立的. 令 $P(x)$ 为需要生成的随机变量的累积分布函数. 令 $Q(x)=P^{-1}(x)$ 为其反函数. 如果 $U[0,1]$ 表示 [0,1] 中的均匀随机数, 则 $Q(U[0,1])$ 便生成了需要的随机变量. 剩余要做的就是找到计算 Q 的尽可能有效的方法.

标准正态分布或高斯分布的随机变量可通过概率分布函数

$$p(x)=\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{x^2}{2}}$$

(即著名的钟形曲线) 来选取实数. 服从分布 $N(0,1)$ 的随机变量的期望为 0, 方差为 1. 更一般地, 服从分布 $N(\mu,\sigma^2)=\mu+\sigma N(0,1)$ 的正态随机变量的期望为 μ, 方

差为 σ^2. 该随机变量只是将标准正态分布 $N(0,1)$ 的随机变量进行了平移和缩放, 在后面部分我们将研究其生成方法.

尽管可以像前面简述的那样直接应用累积分布函数的反函数, 但是同时生成两个正态随机数却更有效. 二维标准正态分布的概率分布函数为 $p(x,y) = (1/2\pi)\mathrm{e}^{-(x^2+y^2)/2}$, 或为极坐标下的 $p(r) = (1/2\pi)\mathrm{e}^{-r^2/2}$. 因为 $p(r)$ 具有极坐标对称性, 所以只需要根据 $p(r)$ 生成径向距离 r, 然后在 $[0,2\pi]$ 之间任选一个角度 θ 即可. 由于 $p(r)$ 对 r^2 服从参数为 $a = 1/2$ 的指数分布, 于是可由 (9.6) 式并通过

$$r^2 = \frac{-\ln(1-u_1)}{1/2}$$

生成 r, 其中 u_1 是均匀随机数. 于是

$$\begin{aligned} n_1 &= r\cos 2\pi u_2 = \sqrt{-2\ln(1-u_1)}\cos 2\pi u_2, \\ n_2 &= r\sin 2\pi u_2 = \sqrt{-2\ln(1-u_1)}\sin 2\pi u_2. \end{aligned} \tag{9.7}$$

为独立正态随机数对, 其中 u_2 为第二个均匀随机数. 注意到在公式中 $1-u_1$ 可由 u_1 代替, 因为分布 $U[0,1]$ 被 1 减后不变. 这就是生成正态随机数的 **Box-Muller 方法**[2]. 这里需要用到平方根、对数、正弦和余弦运算.

当 u_1 用另一种不同的方法生成时, 可以得到一个更有效的 Box-Muller 方法. 从 $U[0,1]$ 中选取 x_1, x_2, 当表达式的值小于 1 时定义 $u_1 = x_1^2 + x_2^2$, 反之将 x_1, x_2 剔除, 重新开始. 可以看到该方法选取的 u_1 服从分布 $U[0,1]$. 该方法的优点在于, 可以使用 $u_2 = \arctan\frac{x_2}{x_1}$, 即从原点到点 (x_1, x_2) 的线段的角度, 显然这是因为 u_2 在 $[0,2\pi]$ 上是均匀分布. 因为 $\cos 2\pi u_2 = \frac{x_1}{u_1}$, $\sin 2\pi u_2 = \frac{x_2}{u_1}$, 所以公式 (9.7) 可转换为

$$n_1 = x_1\sqrt{\frac{-2\ln(u_1)}{u_1}}, \qquad n_2 = x_2\sqrt{\frac{-2\ln(u_1)}{u_1}}, \tag{9.8}$$

其中 $u_1 = x_1^2 + x_2^2$, 这里不需要像 (9.7) 中的正弦和余弦计算.

修正的 Box-Muller 方法是一个**拒绝方法** (rejection method), 因为有些输入值并没有用到. 把单位圆和单位矩形 $[-1,1]\times[-1,1]$ 作比较可知, 进行拒绝运算要多花 $(4-\pi)/4 \approx 21\%$ 的时间. 为避免正弦和余弦计算, 这是一个可接受的代价.

还有许多复杂精巧的方法用于生成正态随机数. 更多的细节可参看 [12]. 例如 MATLAB 中的 **randn** 命令, 使用了 Marsaglia 和 Tsang[15] 的 "ziggurat" 方法, 本质上它是一种有效的反转累积分布函数的方法.

习题 9.1

1. 求由下面条件定义的线性同余生成元的周期:
   (a) $a = 2, b = 0, m = 5$; (b) $a = 4, b = 1, m = 9$.

2. 求由 $a=4, b=0, m=9$ 定义的 LCG 的周期. 该周期是否跟种子有关?
3. 用下面的 LCG 近似求解曲线 $y=x^2(0 \leqslant x \leqslant 1)$ 下方的面积:
   (a) $a=2, b=0, m=5$; (b) $a=4, b=1, m=9$.
4. 由下面的 LCG 生成元近似求解曲线 $y=1-x(0 \leqslant x \leqslant 1)$ 下方的面积:
   (a) $a=2, b=0, m=5$; (b) $a=4, b=1, m=9$.
5. 考虑随机数生成元 RANDNUM-CRAY, 它曾用在第一代超级计算机 Cray X-MP 上. 该 LCG 令 $m=2^{48}, a=2^{24}+3, b=0$. 证明 $u_{i+1}=6u_{i+1}-9u_i(\bmod 1)$. 该生成元存在缺陷吗? 参看计算机问题 9 和计算机问题 10.

计算机问题 9.1

1. 通过执行最小标准随机数生成元, 求例 9.3 中体积的蒙特卡罗近似解. 令种子为 x_0=1, 使用 10^6 个三维点. 该近似解与准确解相差多少?
2. 像问题 1 那样通过执行 randu, 求例 9.3 中体积的蒙特卡罗近似解. 证明任意点 (u_i, u_{i+1}, u_{i+2}) 都不会落在给定的球中.
3. (a) 使用微积分方法, 求由抛物线 $P_1(x)=x^2-x+1/2$ 和 $P_2(x)=-x^2+x+1/2$ 围成的面积. (b) 通过在区间 [0,1] 上求 $P_2(x)-P_1(x)$ 的平均值, 以蒙特卡罗类型 1 问题来求近似面积. (c) 与 (b) 类似, 但以蒙特卡罗类型 2 问题进行估计: 求在矩形 $[0,1]\times[0,1]$ 中落入抛物线之间的点的比例. 比较两种蒙特卡罗方法的效率.
4. 对由多项式 $P_1(x)=x^3$ 和 $P_2(x)=2x-x^2$ 落入第一象限部分的交集, 像计算机问题 3 那样重新求解一次.
5. 用 $n=10^4$ 个伪随机点来估计下面椭圆围成的面积: (a) $13x^2+34xy+25y^2 \leqslant 1$, 其中 $-1 \leqslant x, y \leqslant 1$ (b) $40x^2+25y^2+y+9/4 \leqslant 52xy+14x$, 其中 $0 \leqslant x, y \leqslant 1$. 把估计值与准确值 (a) $\pi/6$ 和 (b) $\pi/18$ 作比较, 并求出误差. 用 $n=10^6$ 重新求解一次并比较结果.
6. 用 $n=10^4$ 个伪随机点来估计由椭球 $2+4x^2+4z^2+y^2 \leqslant 4x+4z+y$ 围成的体积, 其中 $0 \leqslant x, y, z \leqslant 1$. 与准确值 $\pi/24$ 作比较并求出误差. 用 $n=10^6$ 重做一次.
7. (a) 求积分 $\int_0^1 \int_{x^2}^{\sqrt{x}} xy\mathrm{d}x\mathrm{d}y$. (b) 用落入单位正方形 $[0,1]\times[0,1]$ 中的 $n=10^6$ 个数对, 以蒙特卡罗类型 1 问题来估算该积分. (定义一个函数, 当 (x,y) 落入积分区域内时其值为 xy, 落入其他部分时值为 0, 并对该函数求平均值.)
8. 在单位矩形中用 $n=10^6$ 个随机数对估算 $\int_A xy\mathrm{d}x\mathrm{d}y$, 其中区域 A 是由例 9.2 给定的区域.
9. 执行习题 5 中的不可靠的随机数生成元, 并画出类似图 9.3 的图像.
10. 用例 9.3 中的思想, 构造一个蒙特卡罗逼近问题, 使习题 5 中的 RANDNUM-CRAY 生成元完全无效.

9.2 蒙特卡罗模拟

前面已经介绍了蒙特卡罗模拟的两种类型的例子. 本节将研究该技术的适用范围及改进, 包括拟随机数. 本节将用到随机变量和期望的概念.

9.2.1 蒙特卡罗估计的幂定律

我们希望了解蒙特卡罗模拟的收敛速度. 当选取的点 n 变化时, 估计误差变化的速度如何? 这类似于第 5 章中求积方法的收敛性问题, 以及第 6~8 章中微分方程解法的收敛性问题. 在前面的情形中, 提出了关于误差和步长的问题. 在蒙特卡罗模拟中, 减少步长相当于增加随机数.

考虑蒙特卡罗类型 1, 用随机样本计算函数平均值后乘以积分区域的体积得到计算结果. 计算一个函数的平均值可以看作是计算由其决定的概率分布的平均值. 用记号 $\mathrm{E}(X)$ 表示随机变量 X 的期望. 随机变量 X 的**方差**为 $\mathrm{E}[(X-E(X))^2]$, **标准差** (standard deviation) 是方差的平方根. 随着随机点个数 n 的变化, 估计值的误差以下面方式变化:

伪随机数的蒙特卡罗问题类型 1 或类型 2

$$\text{误差} \propto n^{-\frac{1}{2}} \tag{9.9}$$

要理解该式, 可以将积分看作定义域的面积与函数在该定义域上的平均值的乘积. 考虑在一个随机点处的独立同分布变量 X_i, 其平均值就是随机变量 $Y = (X_1+\cdots+X_n)/n$ 的期望, 即

$$\mathrm{E}\left[\frac{X_1+\cdots+X_n}{n}\right] = nA/n = A,$$

Y 的方差为

$$\mathrm{E}\left[\left(\frac{X_1+\cdots+X_n}{n}-A\right)^2\right] = \frac{1}{n^2}\sum \mathrm{E}[(X_i-A)^2] = \frac{1}{n^2}n\sigma^2 = \frac{\sigma^2}{n}$$

其中 σ 是每个 X_i 的标准差. 所以 Y 的标准差减小为 $\sigma/\sqrt{n}$. 该讨论同时适用于蒙特卡罗模拟的类型 1 和类型 2.

亮点 收敛性

蒙特卡罗类型 1 估计与第 5 章中的复合中点方法非常相似. 在那里误差与步长 h 成比例, 约为 $1/n, n$ 函数运算次数. 这要比蒙特卡罗的平方根幂定律更有效.

但是, 蒙特卡罗方法可以求解像例 9.2 那样的问题. 尽管收敛速度仍很慢, 但是还不太清楚如何建立像类型 1 那样的问题从而应用第 5 章的技术.

例 9.4 对曲线 $y=x^2, x\in[0,1]$, 求其下方的面积, 使用伪随机数求蒙特卡罗类型 1 和类型 2 估计.

这是例 9.1 蒙特卡罗类型 1 问题的推广, 那里考虑了当误差作为随机数个数 n 的函数的情况. 对于每次试验, 生成 [0,1] 中的 n 个均匀的随机数 x, 并求出 $y = x^2$ 的平均值. 那里的误差是平均值和准确值 1/3 之间的绝对误差. 对每个 n 做 500 次试验并对误差求平均, 图 9-4 中下面的曲线描绘出了结果.

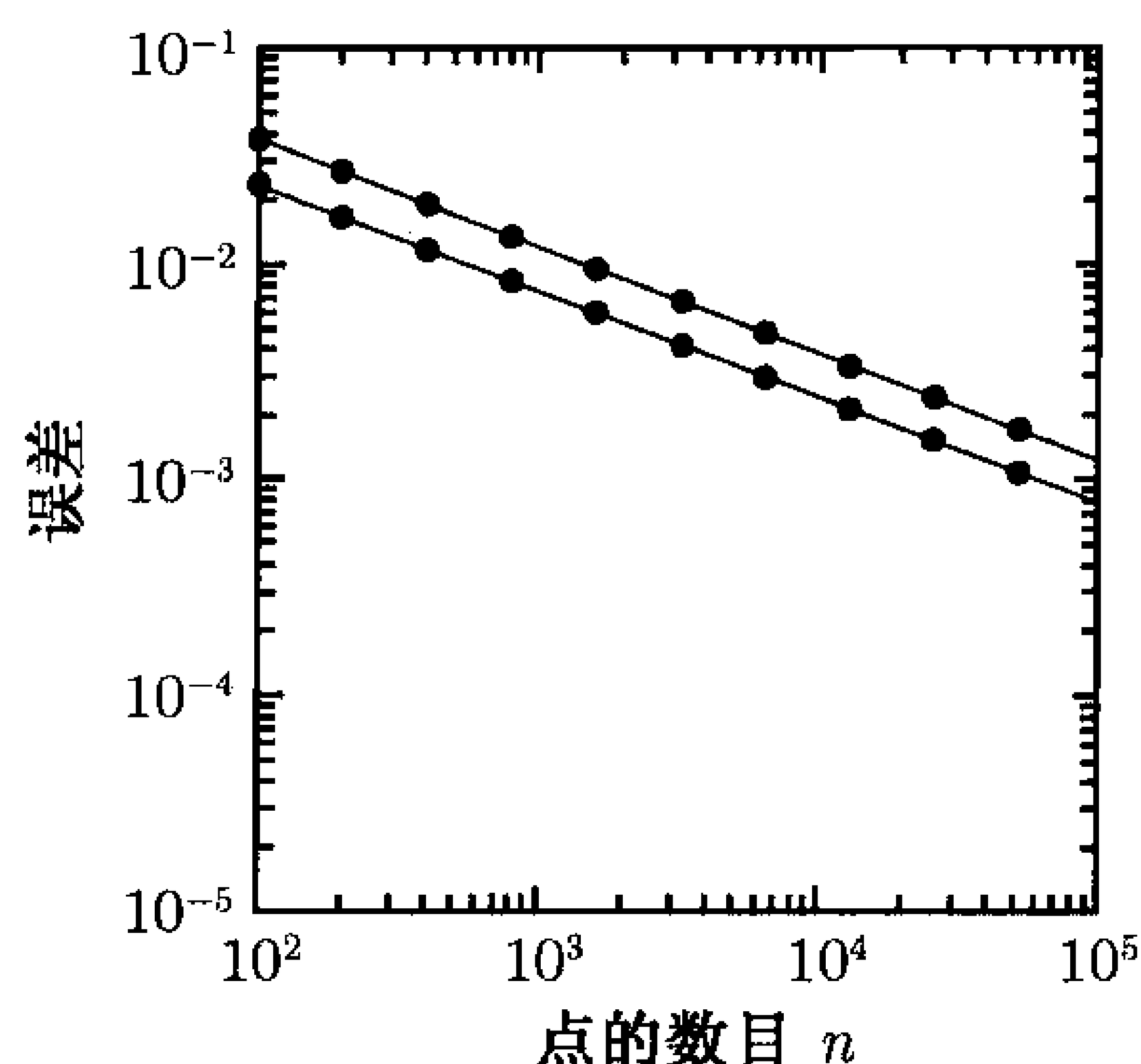


图 9-4 蒙特卡罗估计的平均误差. 例 9.4 中使用伪随机数的蒙特卡罗问题的类型 1 (下面曲线) 和类型 2(上面曲线) 的估计误差. 两种类型的幂定律的指数都是 $-1/2$

对蒙特卡罗类型 2 问题, 在单位正方形 $[0,1] \times [0,1]$ 中生成均匀随机点 (x, y) 并记录满足性质 $y < x^2$ 的点的比例. 同样, 对 500 次试验的误差取平均, 图 9-4 上面的曲线描绘出了结果. 尽管类型 2 的误差要稍大于类型 1 的误差, 但是两者都满足平方根幂定律 (9.9). ◀

对蒙特卡罗类型 2 问题来说, 样本的随机性是必需的吗? 为什么不用样本的矩形式网络取代随机数来解决像例 9.2 中的问题呢? 当然, 对一个任意的样本数 n 是不可行的, 除非可以找到一种看起来随机的方法将它们排序以避免估计中的巨大误差. 确实存在一个中间选择, 可以保持矩形式网格的优点, 但需要将数字重新排序使之看起来像是随机的. 这是 9.2.2 节的主题.

9.2.2 拟随机数

拟随机数 (quasi-random number) 的思想是当独立性对要解决的问题没有本质性的影响时, 牺牲随机数的独立性. 牺牲独立性意味着拟随机数不但不随机, 而且也不需要像伪随机数那样看起来独立. 这样做的目的是为了在蒙特卡罗问题中加快收敛速度. 设计拟随机数序列时需要使之具备 "自避免" (self-avoiding) 的性质而非独立性. 即生成的数列应有效地填充前面数所留出的空隙, 避免与前面已生成的聚在一起. 与伪随机数的比较见图 9-5.

生成拟随机数的方法有很多. 可能最流行的方法要追溯到 1935 年 Van der

Corput 提出的一种称为**基 p 的低歧义数列** (base-p low-discrepancy sequence). 下面给出了基于 Halton[5] 的实施. 令 p 为素数, 例如 $p=2$. 将前 n 个整数写入基 p 中. 假设第 i 个整数具有形式 $b_k b_{k-1}\cdots b_2 b_1$, 则记第 i 个随机数为 $0.b_1 b_2\cdots b_{k-1} b_k$, 并再次写入基 p. 换言之, 将第 i 个整数写入基 p, 然后将其表达式数字反转并写在小数点右边, 得到 [0,1] 中第 i 个均匀随机数. 当 $p=2$ 时表 9-2 给出了前 8 个随机数.

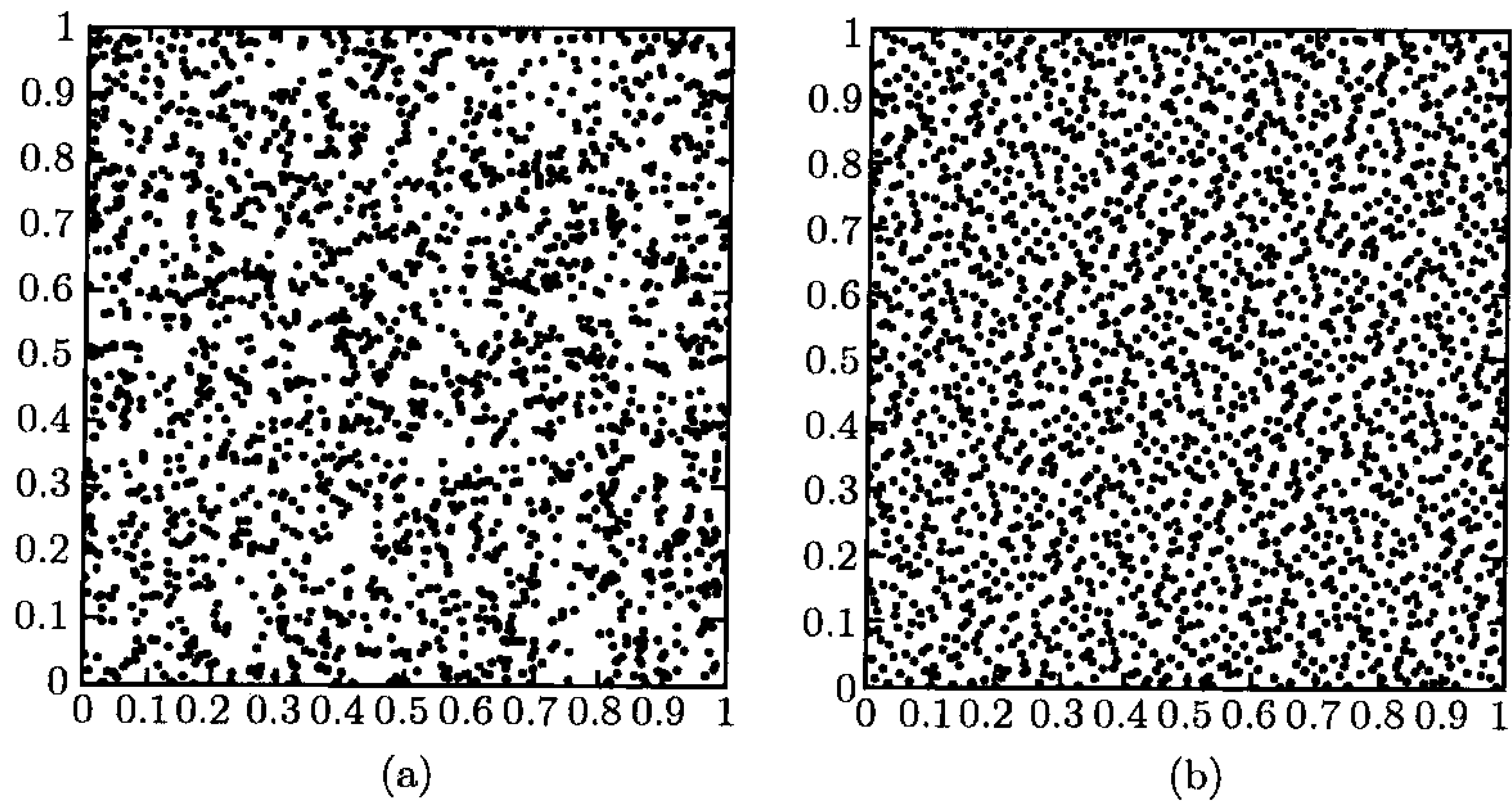


图 9-5　伪随机数与拟随机数的比较. (a) 由 MATLAB 的 rand 命令生成的 2 000 对伪随机数. (b) 由 Halton 低歧义数列生成的 2 000 对拟随机数, x 坐标基为 2, y 坐标基为 3

表　9-2

i	$(i)_2$	$(u_i)_2$	u_i
1	1	0.1	0.5
2	10	0.01	0.25
3	11	0.11	0.75
4	100	0.001	0.125
5	101	0.101	0.625
6	110	0.011	0.375
7	111	0.111	0.875
8	1 000	0.000 1	0.062 5

当 $p=3$ 时得到以下 Hanlton 数, 见表 9-3.

表　9-3

i	$(i)_3$	$(u_i)_3$	u_i
1	1	0.1	$0.\bar{3}$
2	2	0.2	$0.\bar{6}$
3	10	0.01	$0.\bar{1}$
4	11	0.11	$0.\bar{4}$
5	12	0.21	$0.\bar{7}$
6	20	0.02	$0.\bar{2}$
7	21	0.12	$0.\bar{5}$
8	22	0.22	$0.\bar{8}$

下面给出了 Halton 数列的 MATLAB 代码. 它是最初的低歧义思想的简单而直观的版本. 要想更有效, 可以把它写成机器代码.

```
% Program 9.1  Quasi-random number generator
% Halton sequence in base p
% Input: prime number p, random numbers required n
% Output: array u of quasi-random numbers in [0,1]
% Example usage: halton(2,100)
function u=halton(p,n)
b=zeros(ceil(log(n)/log(p)),1);    % largest number of digits
for j=1:n
  i=1;
  b(1)=b(1)+1;                     % add one to current integer
  while b(i)>p-1+eps               % this loop does carrying
    b(i)=0;                        %    in  base p
    i=i+1;
    b(i)=b(i)+1;
  end
  u(j)=0;
  for k=1:length(b(:))             % add up reversed digits
    u(j)=u(j)+b(k)*p^(-k);
  end
end
```

对任意素数, 都可由 Halton 数列得到一个拟随机数集合. 要得到一个 d 维向量的序列, 可以对不同的坐标使用不同的素数. 要记住拟随机数不是独立的, 它们的优点在于它们的自避免性质. 下面将看到, 对于蒙特卡罗问题, 拟随机数要比伪随机数更有效.

用拟随机数的原因是, 对于蒙特卡罗模拟, 拟随机数可以得到更快的收敛速度. 这意味着, 误差作为函数运算次数 n 的函数, 其关于 n 减小速率要快于伪随机数. 下面的公式可以与伪随机数的对应式 (9.9) 作比较 (d 表示随机数的生成维数).

拟随机数的蒙特卡罗问题类型 1

$$\text{误差} \propto (\ln n)^d n^{-1} \tag{9.10}$$

拟随机数的蒙特卡罗问题类型 2

$$\text{误差} \propto n^{-\frac{1}{2}-\frac{1}{2d}} \tag{9.11}$$

误差主要受不连续部分影响. 不加证明, 下面对前面碰到的类型 2 的例子加以说明, 其中函数为具有 $(d-1)$ 维边界的 d 维空间中的子集的特征函数. 在该例子中, 沿着集合边界的不连续点的数目与 $(n^{1/d})^{d-1}$ 成比例. 这与边界是 $(d-1)$ 维的条件一致, 在每个 d 维约有 $n^{1/d}$ 个格点. 这些点随机的取值为 0 或 1, 这取决于该

点在边界的哪一边. 因为在其他点处的误差很小, 关于函数运算的方差平均为

$$\frac{n^{\frac{d-1}{d}}}{n}=n^{-\frac{1}{d}},$$

标准差是其平方根 $n^{-\frac{1}{2d}}$. 对伪随机数的蒙特卡罗问题进行同样的讨论, 对 n 个点进行平均时, 标准差以比率 $\sqrt{n}$ 减少, 这样拟蒙特卡罗方法的标准差为

$$\frac{n^{-1/(2d)}}{n^{1/2}}=n^{-\frac{1}{2}-\frac{1}{2d}}.$$

例 9.5　用拟随机数, 对 $[0,1]$ 中曲线 $y=x^2$ 下方的面积进行蒙特卡罗估计.

这是蒙特卡罗类型 1 问题, 其中 x 坐标可在 [0,1] 中生成, 并求出函数 $f(x)=x^2$ 的平均值来近似求解面积. 用 p=2 的 Halton 数列来生成 10^5 个拟随机数. 这个结果, 以及与使用伪随机数的同样策略结果的比较见图 9-6. 正如前面所说的, 拟随机数明显更有效. ◀

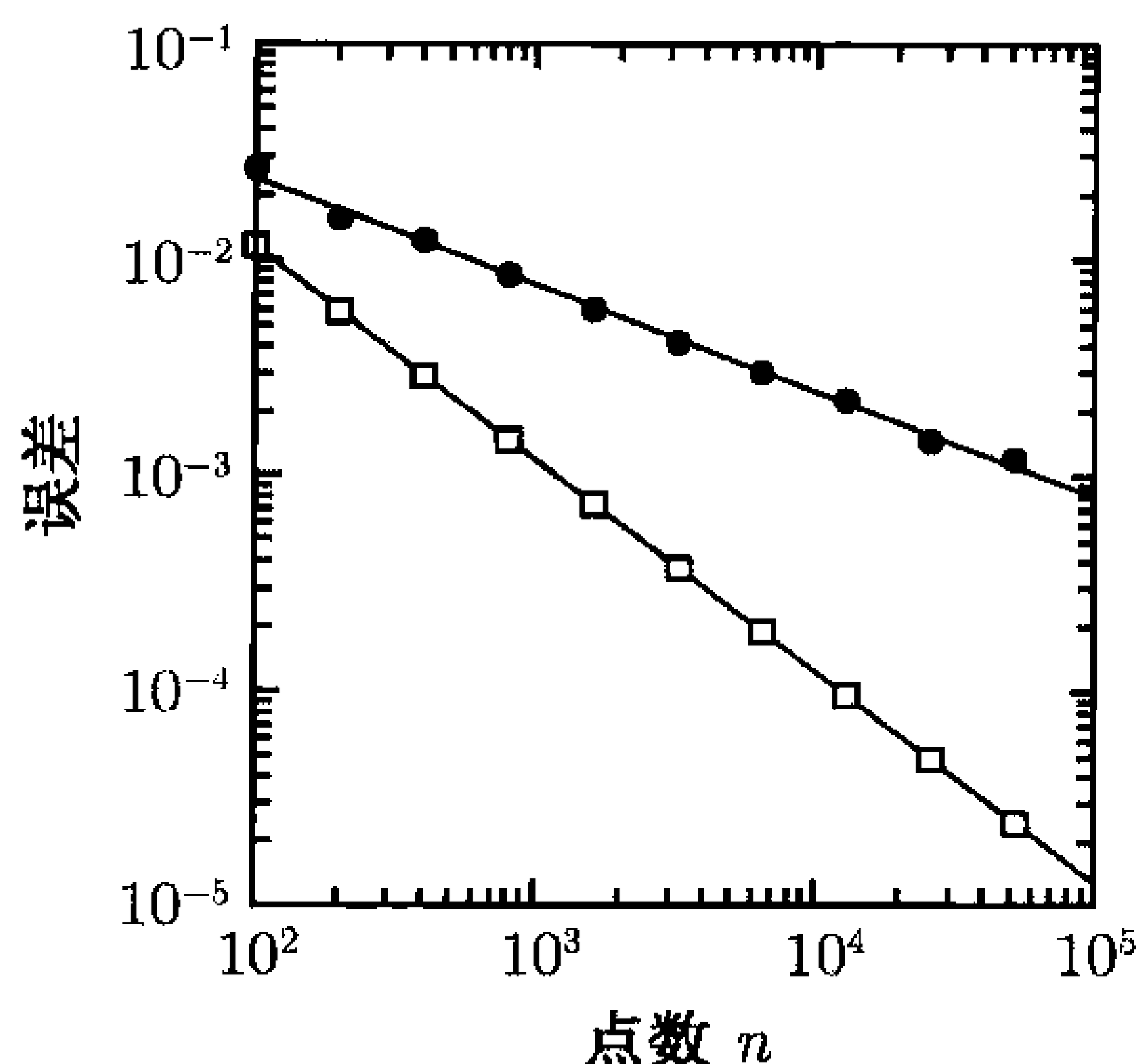


图 9-6　蒙特卡罗类型 1 估计的平均误差. 例 9.1 中积分的估计. 圆点表示使用伪随机数得到的误差, 方块表示用拟随机数得到的误差. 注意到幂定律分别依赖于伪随机数和拟随机数的幂指数 $-1/2$ 和 -1

例 9.6　对例 9.2 中的面积, 用拟随机数求其蒙特卡罗逼近.

对于不同的 n, 单位方块中的拟随机样本由 Halton 数列生成. 对于多维的情况, 对每一坐标使用不同素数 p 的 Halton 数列是方便的. 所求区域是具有一维边界的二维空间中的一个子集, 因此 $d=2$. 确定了满足例 9.2 给定条件的比例, 并计算了误差. 图 9-7a 描绘了 50 次试验的平均误差. 二维蒙特卡罗类型 2 问题的幂定律的指数为 $-1/2-1/(2d)=-1/2-1/4=-3/4$, 即下面曲线的近似斜率. 对伪随机数做同样的计算, 二者的比较如图所示. ◀

例 9.7　用拟随机数的蒙特卡罗估计, 近似求解 R^3 中的半径为 1 的三维球的体积.

类似于例 9.6 那样求解. 因为该类型 2 问题的维数为 3, 所以幂定律的指数为 $-1/2-1/6=-2/3$, 近似等于图 9-7b 中下面的曲线的斜率. ◀

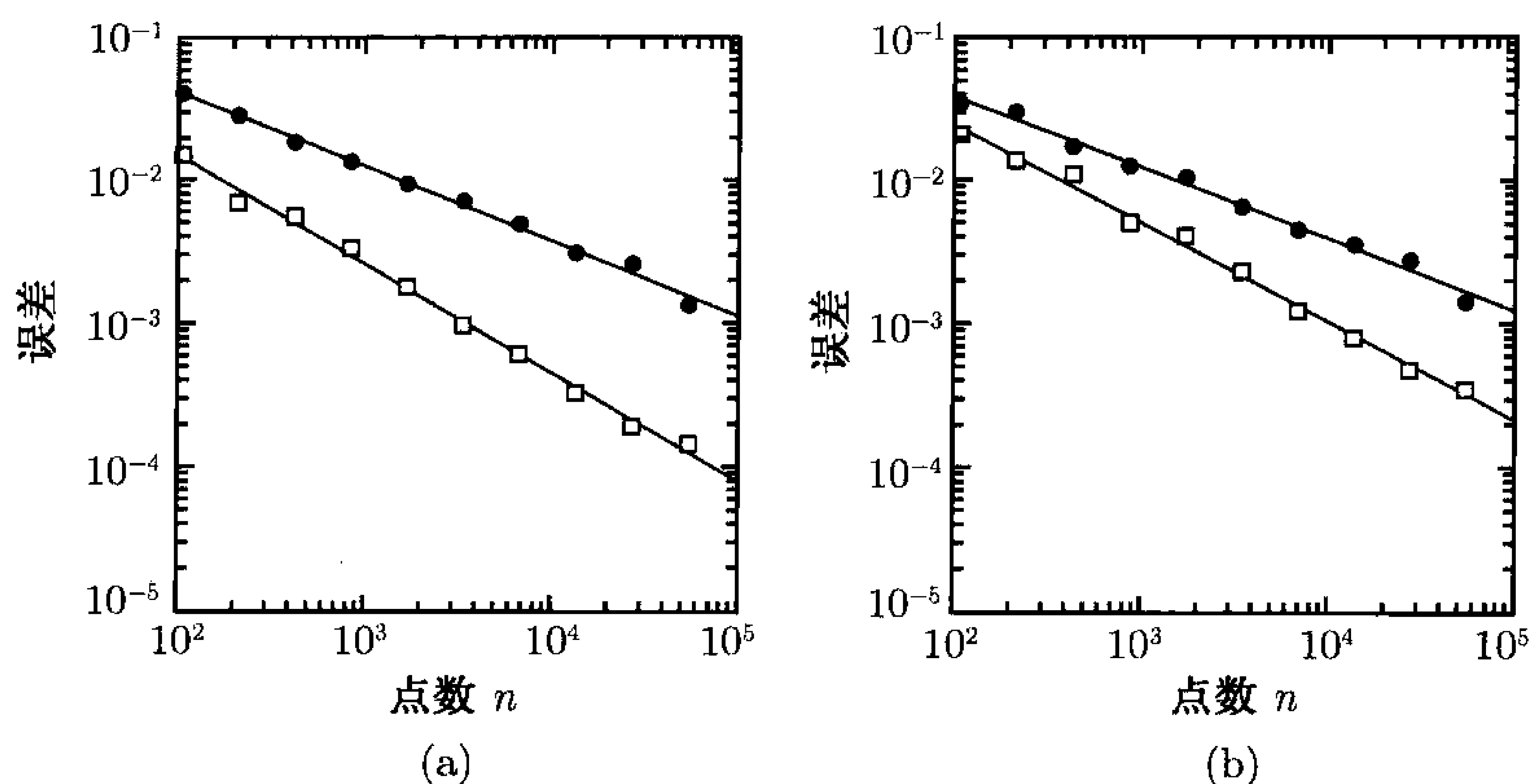


图 9-7 蒙特卡罗类型 2 估计的平均误差. 圆点表示用伪随机数得到的误差, 方块表示用拟随机数得到的误差. (a) 例 9.2 中面积的估计, 属于二维的蒙特卡罗类型 2 问题. 对伪随机数和拟随机数, 误差分别服从指数为 $-1/2$ 和 $-3/4$ 的幂定律. (b) 直径为 1 的三维球的体积的估计, 属于三维蒙特卡罗类型 2 问题. 误差分别服从指数为 $-1/2$ 和 $-3/2$ 的幂定律

计算机问题 9.2

1. 利用由 Halton 数列生成的 $n=10^k$ 个拟随机数, 对计算机问题 9.13 进行蒙特卡罗逼近, 其中 k 分别为 2, 3, 4, 5. 对于小题 (c), 对 x 坐标和 y 坐标分别使用命令 `halton(2,n)` 和 `halton(3,n)`.
2. 用拟随机数对计算机问题 9.1.4 进行蒙特卡罗逼近.
3. 分别用 $n=10^4$ 和 $n=10^5$ 个拟随机数, 对计算机问题 9.1.5 进行蒙特卡罗逼近.
4. 分别用 $n=10^4$ 和 $n=10^5$ 个拟随机数, 对计算机问题 9.1.6 进行蒙特卡罗逼近.
5. 对半径为 1 的四维球的体积, 用 $n=10^5$ 个点分别求其蒙特卡罗逼近和拟蒙特卡罗逼近. 并与精确体积 $\pi^2/2$ 进行比较.
6. 一个最有名的蒙特卡罗问题就是 Buffon 针问题. 如果一个针掉到一个有黑白条纹的平面上, 条纹的宽度与针的长度相同, 则该针跨两种颜色的条纹的概率为 $2/\pi$. (a) 从分析上证明这个结果. 考虑针的中点到最近的条纹边缘的距离 d 和其与条纹所成的角度 θ. 把概率表示为一个简单的积分. (b) 设计一个蒙特卡罗类型 2 模拟来近似求解该概率, 使用 $n=10^6$ 个伪随机数对 (d,θ).
7. (a) 元素在区间 [0,1] 上取值的 2×2 矩阵中, 正定矩阵占的比例是多少? 求出精确值, 并用蒙特卡罗模拟逼近. (b) 元素在区间 [0,1] 上取值的 2×2 对称矩阵中, 正定矩阵占的比例是多少? 求出精确解, 并用蒙特卡罗模拟逼近.
8. 对元素在区间 $[-1,1]$ 上取值的 2×2 矩阵, 特征值都为实数的矩阵所占的比例是多少, 用蒙特卡罗模拟近似求解.

9. 元素在区间 [0,1] 上取值的 4×4 矩阵中, 采用部分选主元法没有行交换的矩阵比例有多少? 用蒙特卡罗模拟以及 MATLAB 的 `lu` 命令来估算该概率.

9.3 离散布朗运动和连续布朗运动

尽管本书前面的章节主要研究了对确定性模型很重要的一些原理, 但这些模型只是现代技术中的一部分. 随机数最重要的应用之一就是使得随机建模成为可能.

下面首先阐述最简单的随机模型之一: 随机游动 (也称为离散布朗运动). 离散随机模型的基本原理与后面更复杂的连续布朗运动的基本原理是一样的.

9.3.1 随机游动

随机游动 (random walk) W_t 定义在实轴上, 从 $W_0=0$ 开始, 在每个整数时间 i 移动步长 s_i, 其中 s_i 是独立同分布的随机变量. 这里假设 s_i 以等概率取 $+1$ 和 -1. **离散布朗运动**定义为由下面累积步序列确定的随机游动:

$$W_t = W_0 + s_1 + s_2 + \cdots + s_t, \qquad t = 0, 1, 2, \cdots$$

图 9-8 给出了离散布朗运动的单个实现.

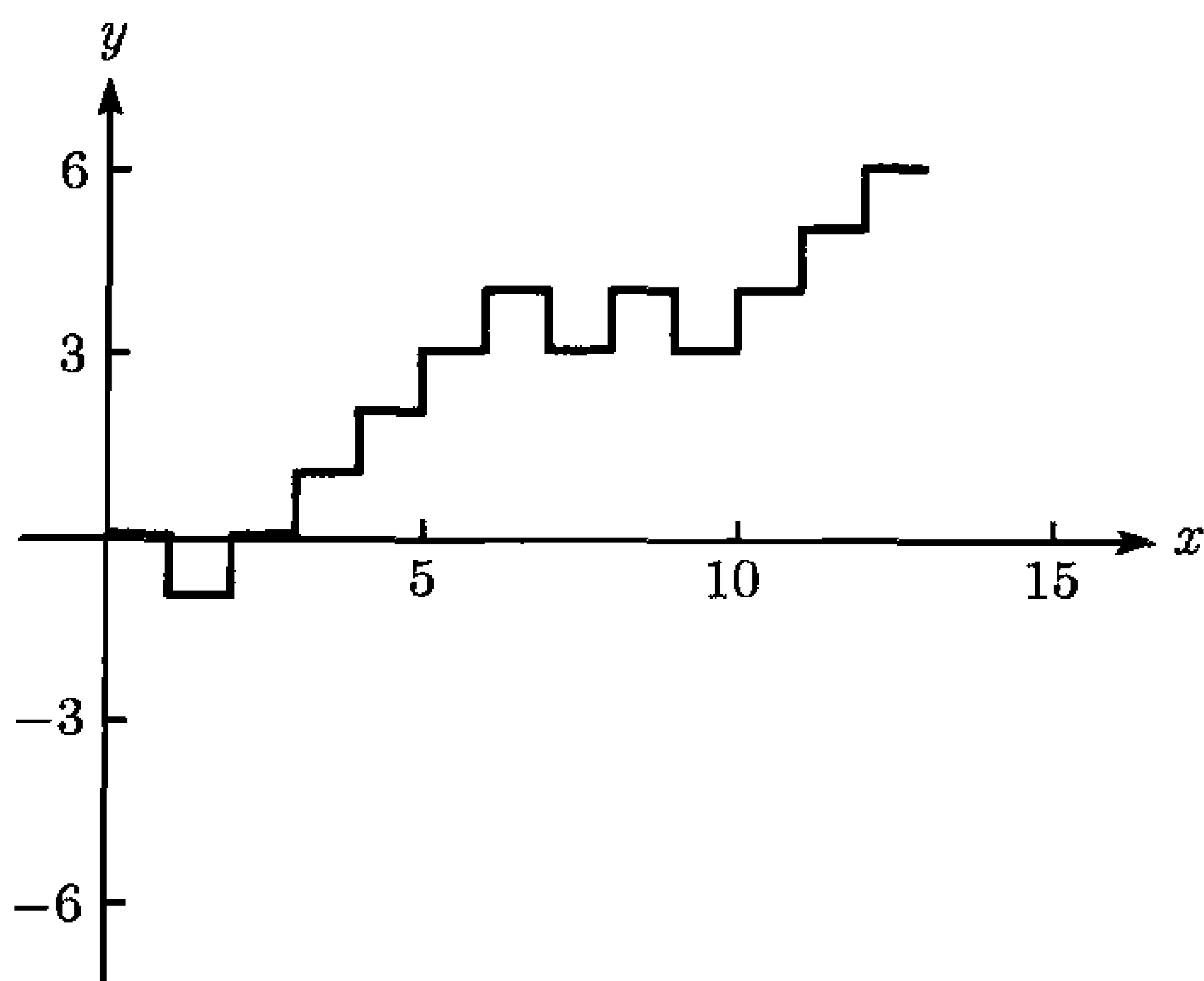


图 9-8 随机游动的单个实现. 在第 12 步时达到了 (垂直) 区间 $[-3,6]$ 的边界. 随机游动超出该区间的最高点需要的平均次数是总次数的 1/3

下面的 MATLAB 代码执行了 10 步随机游动:

```
t=10;
w=0;
for i=1:t
  if rand>1/2
    w=w+1;
  else
    w=w-1;
```

```
    end
end
```

随机游动是一种概率行为, 下面需要一些概率论的基本概念. 对每个 t, W_t 是一个随机变量. 把一些随机变量 $\{W_0, W_1, W_2, \cdots\}$ 串联在一起定义为**随机过程** (stochastic process).

随机游动 W_t 的一个单步 s_i 的期望值为 $(0.5)(1)+(0.5)\times(-1)=0, s_i$ 的方差为 $\mathrm{E}[(s_i-0)^2]=(0.5)(1)^2+(0.5)(-1)^2=1$. 进行 t 步后的随机游动的期望为 $\mathrm{E}(W_t)=\mathrm{E}(s_1+s_2+\cdots+s_t)=\mathrm{E}(s_1)+\mathrm{E}(s_2)+\cdots+\mathrm{E}(s_t)=0$, 方差为 $\mathrm{Var}(W_t)=\mathrm{Var}(s_1+s_2+\cdots+s_t)=\mathrm{Var}(s_1)+\mathrm{Var}(s_2)+\cdots+\mathrm{Var}(s_t)=t$, 因为方差对于相互独立的随机变量是可加的.

期望和方差是描述概率分布的统计量. 由 W_t 的期望为 0, 方差为 t 可知, 要计算随机变量 W_t 的 n 个不同实现, 则

$$\text{样本平均} = \mathrm{E}_{\text{样本}}(W_t) = \frac{W_t^1+\cdots+W_t^n}{n},$$

$$\text{样本方差} = \mathrm{Var}_{\text{样本}}(W_t) = \frac{(W_t^1-E_s)^2+\cdots+(W_t^n-E_s)^2}{n-1}$$

分别趋于 0 和 t. **样本标准差**定义为样本方差的平方根, 也称为平均值的**标准误差** (standard error).

随机游动的许多有趣的应用是基于溢出时间 (escape time), 也称为第一通过时间 (first passage time). 令 a, b 为整数, 考虑随机游动从 0 开始到达 $[-b, a]$ 边界的第一个时间. 这称为随机游动的**溢出时间**. 从 [20] 可知逃逸发生在 a (而非 $-b$) 的概率恰为 $b/(a+b)$.

例 9.8 用蒙特卡罗模拟, 近似求随机游动在边界点 6 溢出区间 $[-3, 6]$ 的概率.

这发生的次数为总次数的 1/3. 作为蒙特卡罗类型 2 问题, 下面计算样本均值和从 $a=6$ 处溢出的概率的误差. 做 n 步随机游动直到溢出, 并记录在到达 -3 前到达 6 的比例. 对 n 的不同取值, 见表 9-4.

表 9-4

n	从上溢出	概率	误差
100	35	0.350 0	0.106 7
200	72	0.360 0	0.026 7
400	135	0.337 5	0.004 2
800	258	0.322 5	0.010 8
1 600	534	0.330 6	0.002 7
3 200	1 096	0.342 5	0.009 2
6 400	2 213	0.345 8	0.012 4

误差是估计值和准确值 1/3 的差的绝对值. 当随机游动步数增加时误差通常应慢慢减小, 但事实并非完全这样, 如表所示. 图 9-9 给出了 50 次试验的平均误差. 从中可以看出, 误差表明以平方根速度下降的幂定律是蒙特卡罗模拟的一个特征. ◀

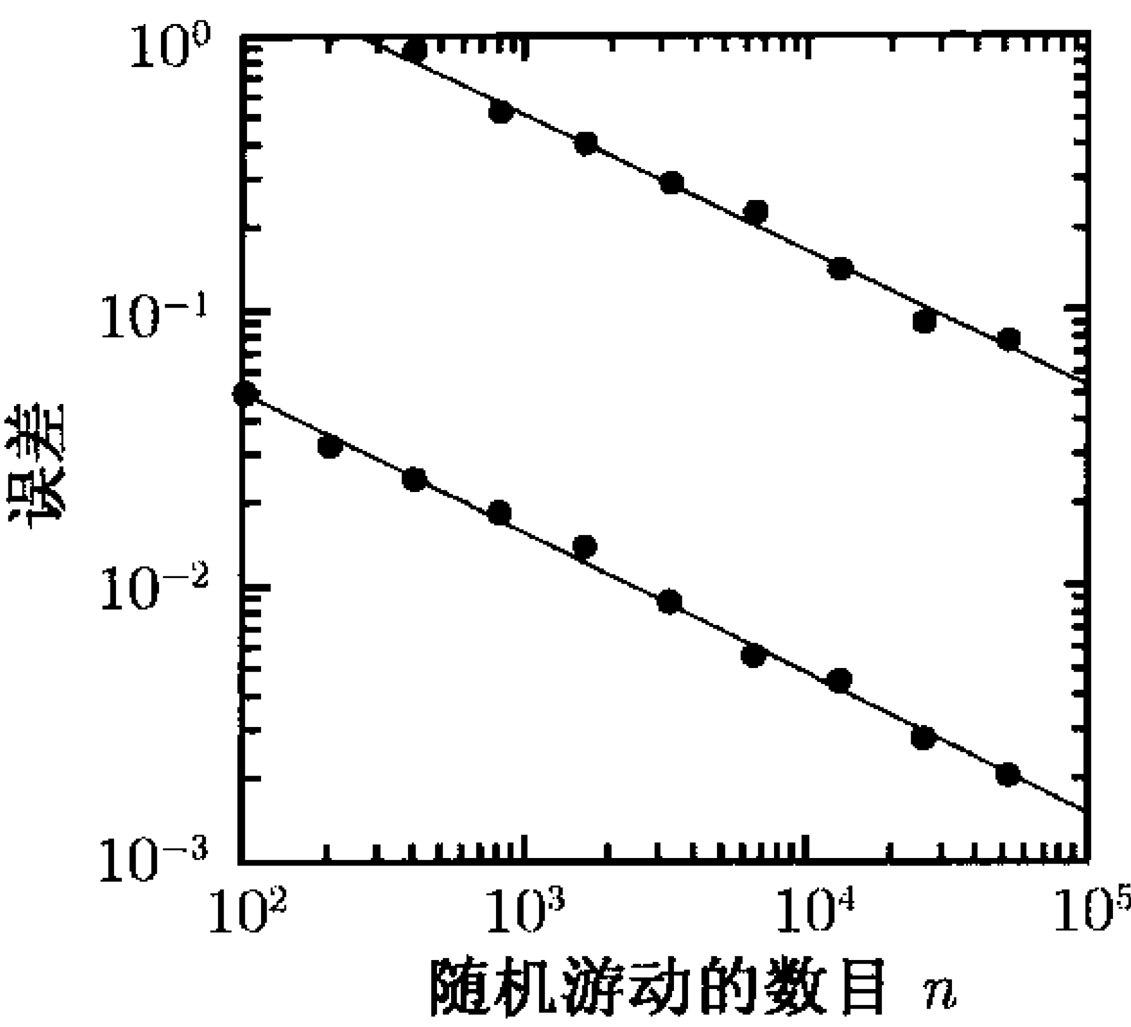


图 9-9 对溢出问题, 蒙特卡罗模拟的误差. 对于从点 6 处溢出区间 $[-3,6]$ 的概率, 估计误差与随机游动步数的关系如下面的曲线所示. 概率的期望值为 1/3. 上面的曲线是同一个问题的溢出时间的估计误差. 期望值为 18 个时间步. 误差为 50 次试验的平均值

从 $[-b,a]$ 溢出时间的期望值为 ab, 见 [20]. 可以用同样的模拟来研究蒙特卡罗模拟在该问题上的有效性.

例 9.9 用蒙特卡罗模拟, 估计随机游动溢出区间 $[-3,6]$ 的溢出时间.

溢出时间的期望值为 $ab=18$. 一个算例给出了表 9-5. 同样, 误差的下降没有固定的速率. 要看误差的平方根幂定律, 对每个 n 要做更多试验取平均. 50 次试验的结果如图 9-9 所示. ◀

表 9-5

n	平均溢出时间	误差
100	18.84	0.84
200	17.47	0.53
400	19.64	1.64
800	18.53	0.53
1 600	18.27	0.27
3 200	18.16	0.16
6 400	18.05	0.05

9.3.2 连续布朗运动

在前面几节中, 在 t 时刻的标准随机游动的期望为 0, 方差为 t. 现在假设单位时间的步数增大 1 倍. 如果每 1/2 时间单位进行一步游动, 则随机游动在 t 时刻的期望为 0, 方差变为

$$\mathrm{Var}(W_t) = \mathrm{Var}(s_1 + \cdots + s_{2t}) = \mathrm{Var}(s_1) + \cdots + \mathrm{Var}(s_{2t}) = 2t,$$

因为有 $2t$ 步游动发生. 像微分方程一样, 为把噪声加到连续模型中, 需要随机游动的连续性版本. 每单位时间使步数增加一倍是一个好的开始, 但为保持方差不变, 需要减小每步的 (纵向) 步长. 如果步数以因子 k 增加, 为保持方差不变则需要以因子 $1/\sqrt{k}$ 减小步长高度. 这是因为若随机变量乘以一个常数, 则其方差要变化该常数的平方倍.

因此, 定义 W_t^k 为每一步 s_i^k 的水平步长是 $1/k$、垂直步长为等概率的 $\pm 1/\sqrt{k}$ 的随机游动. 则在 t 时刻的期望值为

$$\mathrm{E}(W_t^k) = \sum_{i=1}^{kt} \mathrm{E}(s_i^k) = \sum_{i=1}^{kt} 0 = 0,$$

方差为

$$\mathrm{Var}(W_t^k) = \sum_{i=1}^{kt} \mathrm{Var}(s_i^k) = \sum_{i=1}^{kt} \left[\left(\frac{1}{\sqrt{k}}\right)^2 (0.5) + \left(-\frac{1}{\sqrt{k}}\right)^2 (0.5) \right] = kt\frac{1}{k} = t. \quad (9.12)$$

如果随着 k 增加而减少随机游动的水平步长和垂直步长, 则方差和标准差保持为常数, 独立于每单位时间的步数 k. 图 9-10b 给出了 W_t^k 的实现, 其中 $k = 25$, 因此在 10 个单位时间中游动了 250 步. 在 $t = 10$ 时刻的期望和方差与图 9-10a 一样.

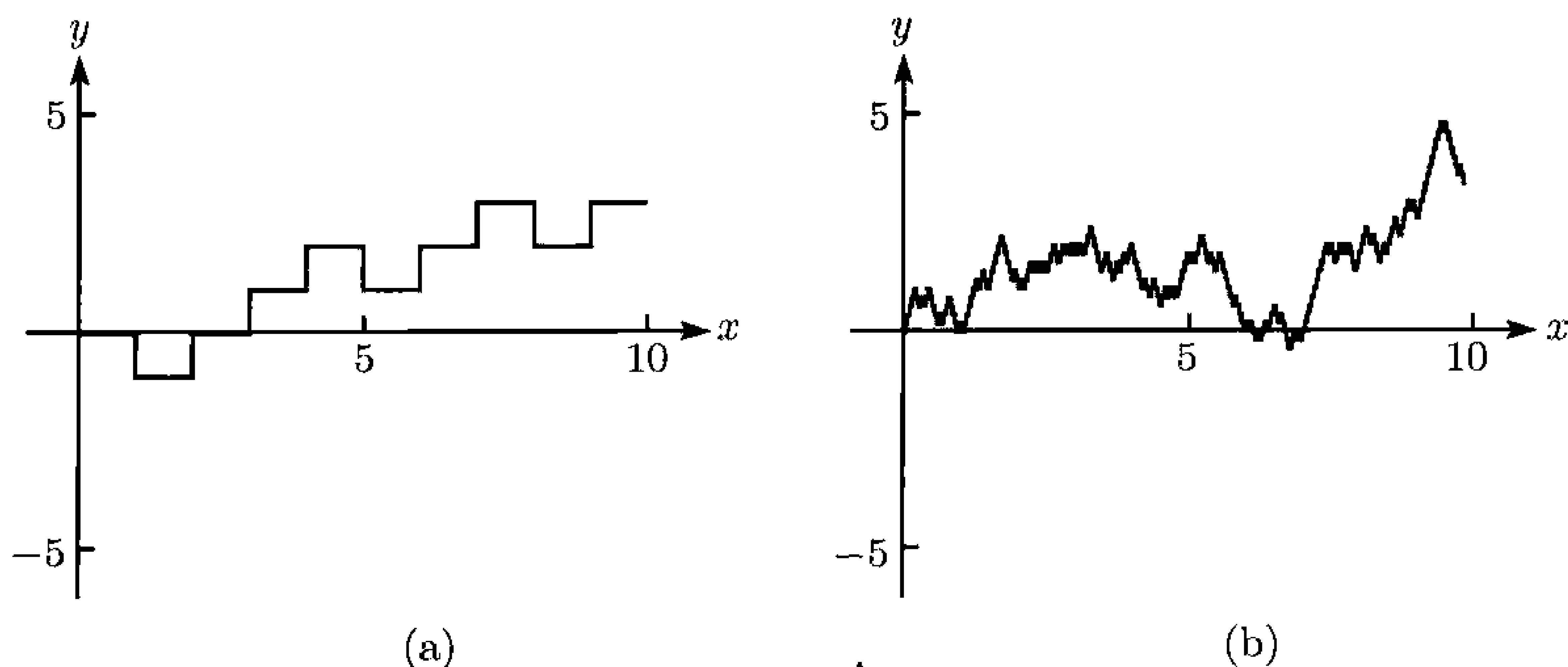


图 9-10 离散布朗运动. (a) 10 步随机游动. (b) 随机游动 W_t^{25} 走的步数是 (a) 的 25 倍, 但是垂直步长是 $1/\sqrt{25}$. 在时刻 $t = 10$ 时, 游动过程 (a) 和 (b) 的期望和方差相同 (分别为 0 和 10)

令 $k \to \infty$, 这个级数的极限 W_t^∞ 便形成了**连续布朗运动**. 这里时刻 t 是个实数, $B_t \equiv W_t^\infty$ 对每个 $t \geqslant 0$ 为随机变量. 连续布朗运动 B_t 有 3 个重要性质.

性质 1 对每个 t, 随机变量 B_t 服从期望为 0、方差为 t 的正态分布.

性质 2 对每个 $t_1 < t_2$, 正态随机变量 $B_{t_2} - B_{t_1}$ 独立于随机变量 B_{t_1}, 并且事实上独立于所有的 B_s, 其中 $0 \leqslant s \leqslant t_1$.

性质 3　*布朗运动 B_t 可用连续轨迹表示.*

正态分布的结果可由中心极限定理得到, 该定理是概率论的一个深刻事实.

布朗运动的计算机模拟以这 3 个性质为基础. 在 t 轴上建立划分

$$0 = t_0 \leqslant t_1 \leqslant \cdots \leqslant t_n$$

从 $B_0 = 0$ 开始. 性质 2 表明增量 $B_{t_1} - B_{t_0}$ 是一个正态随机变量, 期望和方差分别为 0 和 t_1. 因此可以利用正态分布 $N(0, t_1) = \sqrt{t_1 - t_0}N(0, 1)$ 使随机变量 B_{t_1} 得到实现; 换言之, 用标准正态随机数乘以 $\sqrt{t_1 - t_0}$. 类似地可以得到 B_{t_2}. $B_{t_2} - B_{t_1}$ 的分布是 $N(0, t_2 - t_1) = \sqrt{t_2 - t_1}N(0, 1)$, 因此可以选取标准正态随机数, 乘以 $\sqrt{t_2 - t_1}$, 加到 B_{t_1} 上得到 B_{t_2}. 一般地, 布朗运动的增量是时间步长的平方根再乘以一个标准正态随机数.

在 MATLAB 中, 可以利用内置的正态随机数生成元 randn 来逼近布朗运动. 这里使用时间步长 $\Delta t = 1/25$, 如图 9-10b 所示.

```
k=250;
sqdelt=sqrt(1/25);
b=0;
for i=1:k
  b=b+sqdelt*randn;
end
```

连续布朗运动溢出时间的统计量与随机游动的统计量相同. 设 a, b 为正数 (不一定是整数), 考虑连续布朗运动从 0 开始到达区间 $[-b, a]$ 的边界的第一时间. 这称为布朗运动从该区间的溢出时间. 可以证明, 其从 a (而非 $-b$) 溢出的概率为 $b/(a+b)$. 并且溢出时间的期望为 ab. 计算机问题 5 要求读者用蒙特卡罗模拟描述该事实.

计算机问题 9.3

1. 设计一个蒙特卡罗模拟来估算随机游动到达区间 $[-b, a]$ 的顶部 a 的概率. 取 $n = 10\,000$ 步随机游动. 与准确解比较并计算误差. (a) $[-2, 5]$; (b) $[-5, 3]$; (c) $[-8, 3]$.
2. 计算问题 1 中随机游动的平均溢出时间. 取 $n = 10\,000$ 步随机游动. 与准确解比较并计算误差.
3. 在**有偏随机游动** (biased random walk) 中, 向上走一个单位的概率为 $0 < p < 1$, 向下走一个单位的概率为 $q = 1 - p$. 对于计算机问题 1, 设计一个蒙特卡罗模拟, 估算其到达区间顶部的概率, 其中 $n = 10\,000$, $p = 0.7$. 与准确解 $[(q/p)^b - 1]/[(q/p)^{a+b} - 1](p \neq q)$ 比较并计算误差.
4. 对问题 3 的溢出时间重新计算一次. 该有偏随机游动的平均溢出时间 $(p \neq q)$ 为 $[b - (a+b)(1 - (q/p)^b/(1 - (q/p)^{a+b})]/[q - p]$.
5. 设计一个蒙特卡罗模拟, 估算随机游动溢出区间 $[-b, a]$ 的概率. 取 $n = 1\,000$ 步布朗运动, 步长 $\Delta t = 0.01$. 与准确解 $b/(a+b)$ 比较并计算误差. (a) $[-2, 5]$; (b) $[-2, \pi]$;

(c) $[-8/3, 3]$

6. 对计算机问题 5 的区间, 计算布朗运动的平均溢出时间. 取 $n = 1\ 000$ 步布朗运动, 步长 $\Delta t = 0.01$. 与准确解比较并计算误差.

7. 布朗运动的反正弦定律成立, 对 $0 \leqslant t_1 \leqslant t_2$, 运动轨迹在区间 $[t_1, t_2]$ 没有穿越 0 的概率为 $(2/\pi)\arcsin\sqrt{t_1/t_2}$. 对 $\Delta t = 0.01$, 设计一个使用 10 000 个轨迹的蒙特卡罗模拟来估算该概率, 并与准确概率比较, 时间区间分别为: (a) $3 < t < 5$ (b) $2 < t < 10$ (c) $8 < t < 10$.

9.4 随机微分方程

常微分方程 (ODE) 是确定性模型. 给定一个 ODE 以及合适的初值条件, 问题具有唯一解, 即关于方程解的将来的变化是完全确定的. 但该性质对模型并不是总是成立的. 对于许多系统, 尽管有些部分很容易建模, 但是其他部分则可能具有随机性——就好像独立于当前系统的状态. 在这种情况下, 通常把噪声项加入微分方程来表示随机部分, 而不是放弃该模型的使用. 这即称为随机微分方程 (SDE).

本节将讨论一些基本的随机微分方程以及如何用数值方法逼近方程的解. 解为连续随机过程, 类似于布朗运动. 下面从一些必要的定义和 Ito 微积分的简单介绍开始. 更详细的内容, 请参考 [9, 17, 20].

9.4.1 将噪声引入微分方程

常微分方程的解是函数. 而随机微分方程的解为随机过程.

定义 9.2 附以实数 $t \geqslant 0$ 的随机变量 x_t 的集合, 称为**连续时间随机过程**.

每个实例, 或者随机过程的实现, 是由随机变量 x_t 对 t 的一个选择, 即为 t 的函数.

布朗运动 B_t 是一个随机过程. 任何 (确定性) 函数 $f(t)$ 也可以看作是一个平凡的随机过程, 其方差 $\mathrm{Var}(f(t)) = 0$. SDE 初值问题

$$\begin{cases} \mathrm{d}y = r\mathrm{d}t + \sigma\mathrm{d}B_t \\ y(0) = 0 \end{cases} \qquad (\text{其中 } r \text{ 和 } \sigma \text{ 为常数}) \tag{9.13}$$

的解为随机过程 $y(t) = rt + \sigma B_t$, 不过需要限定一些项.

注意到 SDE(9.13) 可以具有不同的形式, 不像 ODE 的衍生形式. 这是因为许多有趣的随机过程是连续但不可微的, 例如布朗运动. 所以由定义, SDE

$$\mathrm{d}y = f(t, y)\mathrm{d}t + g(t, y)\mathrm{d}B_t$$

意味着下述积分方程

$$y(t) = y(0) + \int_0^t f(s, y)\mathrm{d}s + \int_0^t g(s, y)\mathrm{d}B_s,$$

这里仍然把最后一个积分定义为 Ito 积分.

令 $a = t_0 < t_1 < \cdots < t_{n-1} < t_n = b$ 是区间 $[a, b]$ 的一个划分. 黎曼积分定义为极限

$$\int_a^b f(t)\mathrm{d}t = \lim_{\Delta t \to 0} \sum_{i=1}^{n} f(t_i')\Delta t_i,$$

其中 $\Delta t_i = t_i - t_{i-1}$,$t_{i-1} \leqslant t_i' \leqslant t_i$. 类似地, **Ito 积分**定义为极限

$$\int_a^b f(t)\mathrm{d}B_t = \lim_{\Delta t \to 0} \sum_{i=1}^{n} f(t_{i-1})\Delta B_i,$$

其中 $\Delta B_i = B_i - B_{i-1}$, 是一步布朗运动. 虽然黎曼积分中的 t_i' 可以选取为 (t_{i-1}, t_i) 中的任一点, 但是 Ito 积分中相应的点需要取为区间的左端点.

因为 f 和 B_t 是随机变量, 因此 Ito 积分 $I = \int_a^b f(t)\mathrm{d}B_t$ 也是随机变量. **微分** $\mathrm{d}I$ 是为了记号的方便, 因此由定义,

$$I = \int_a^b f\mathrm{d}B_t$$

等价于

$$\mathrm{d}I = f\mathrm{d}B_t.$$

布朗运动 B_t 的微分 $\mathrm{d}B_t$ 称为**白噪声** (white noise).

例 9.10 求解随机微分方程 $y(t) = r\mathrm{d}t + \sigma\mathrm{d}B_t$, 初始条件为 $y(0) = y_0$.

假设 r 和 σ 为常实数. (确定性) 常微分方程

$$y'(t) = r \tag{9.14}$$

的解 $y(t) = y_0 + rt$, 是关于时间 t 的线性函数. 如果 r 为正, 解以常数斜率上升; 如果 r 为负, 那么解就下降.

把带有常实数 σ 的白噪声 $\sigma\mathrm{d}B_t$ 加到等式右边得到随机微分方程

$$\mathrm{d}y(t) = r\mathrm{d}t + \sigma\mathrm{d}B_t. \tag{9.15}$$

两边积分得

$$y(t) - y(0) = \int_0^t \mathrm{d}y = \int_0^t r\mathrm{d}s + \int_0^t \sigma\mathrm{d}B_s = rt + \sigma B_t.$$

这表明, 解为随机过程

$$y(t) = y_0 + rt + \sigma B_t, \tag{9.16}$$

这是漂移 (rt 项) 和布朗运动扩散的组和.

图 9-11 给出了 SDE (9.15) 在 ODE (9.14) 的唯一解附近的两个解. 严格地讲, ODE (9.14) 的解也是 (9.15) 的一个解, 其噪声为 z_i=0. 这是随机过程的一个可能的但是极不可能的特殊解. ◀

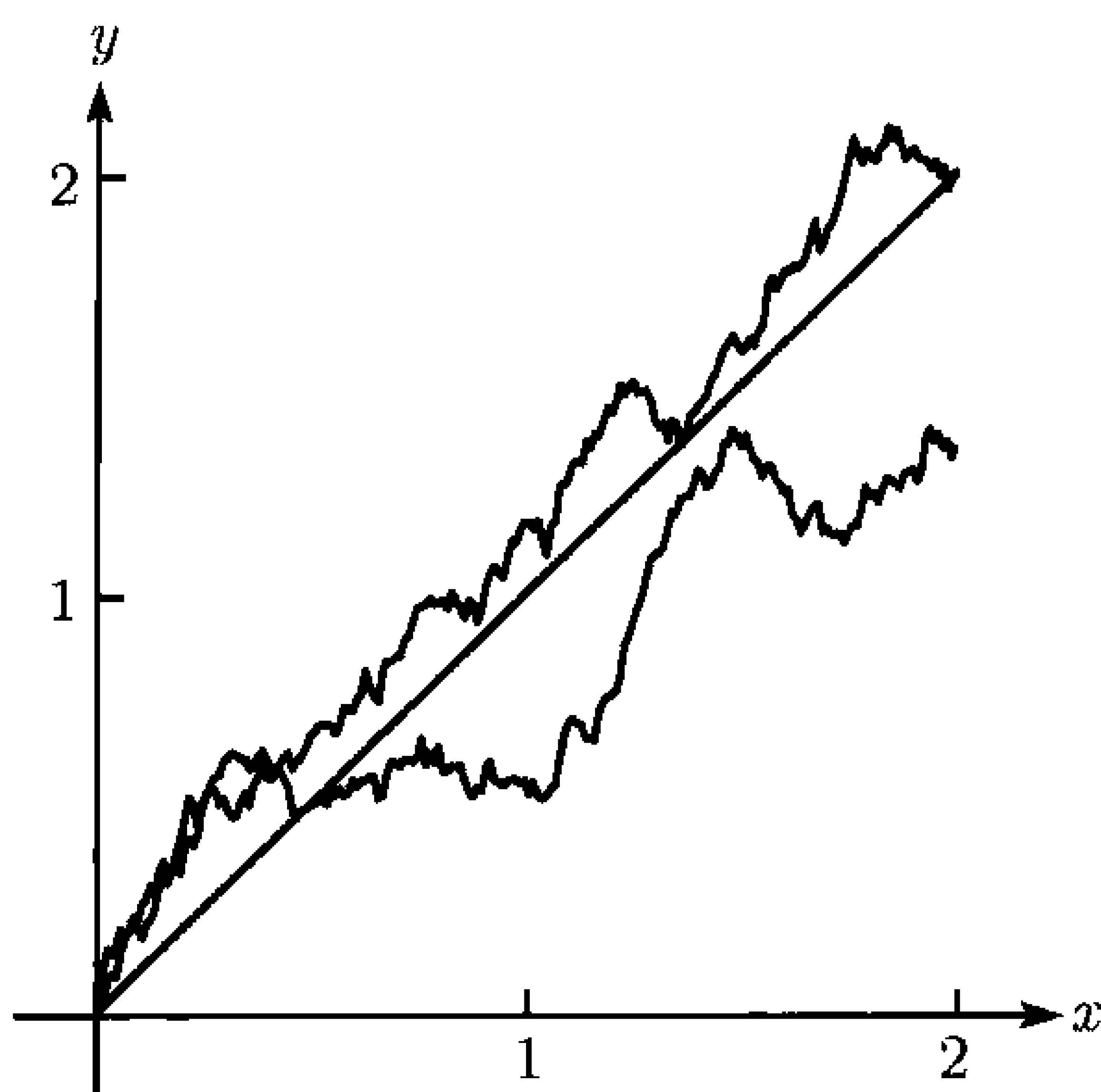


图 9-11 例 10 的解. 图中给出了 ODE $y'(t) = r$ 的一个解 $y(t) = rt$, 以及求解 (9.15) 的随机过程 $y(t) = rt + \sigma B_t$ 的两种实现. 参数 $r = 1$, $\sigma = 0.3$

为从分析上求解 SDEs, 需要介绍一些基本的随机微分的运算法则, 称为 Ito 公式.

Ito 公式

如果 $y = f(t, x)$, 则

$$\mathrm{d}y = \frac{\partial f}{\partial t}(t,x)\mathrm{d}t + \frac{\partial f}{\partial x}(t,x)\mathrm{d}x + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(t,x)\mathrm{d}x\mathrm{d}x, \tag{9.17}$$

其中 $\mathrm{d}x\mathrm{d}x$ 项可由恒等式 $\mathrm{d}t\mathrm{d}t = 0, \mathrm{d}t\mathrm{d}B_t = \mathrm{d}B_t\mathrm{d}t = 0$ 和 $\mathrm{d}B_t\mathrm{d}B_t = \mathrm{d}t$ 来解释.

Ito 公式是传统微积分学中链式法则的随机形式. 虽然该公式用了微分的表达形式, 但是其含义等价于将公式两边 Ito 积分. 由 Ito 积分的定义可以证明该公式 [17].

例 9.11 证明 $y(t) = B_t^2$ 是 SDE$\mathrm{d}y = \mathrm{d}t + 2\mathrm{d}B_t\mathrm{d}B_t$ 的一个解.

为利用 Ito 公式, 记 $y = f(t, x)$, 其中 $x = B_t, f(t, x) = x^2$. 由 (9.17) 式得

$$\begin{aligned}\mathrm{d}y &= f_t\mathrm{d}t + f_x\mathrm{d}x + \frac{1}{2}f_{xx}\mathrm{d}x\mathrm{d}x = 0\mathrm{d}t + 2x\mathrm{d}x + \frac{1}{2}2\mathrm{d}x\mathrm{d}x \\ &= 2B_t\mathrm{d}B_t + \mathrm{d}B_t\mathrm{d}B_t = 2B_t\mathrm{d}B_t + \mathrm{d}t.\end{aligned}$$ ◀

例 9.12 指出几何布朗运动

$$y(t) = y_0\mathrm{e}^{(r-\frac{1}{2}\sigma^2)t+\sigma B_t} \tag{9.18}$$

满足随机微分方程

$$\mathrm{d}y = ry\mathrm{d}t + \sigma y\mathrm{d}B_t. \tag{9.19}$$

记 $y = f(t,x) = y_0\mathrm{e}^x$, 其中 $x = \left(r - \frac{1}{2}\sigma^2\right)t + \sigma B_t$. 由 Ito 公式的定义,

$$\mathrm{d}y = y_0\mathrm{e}^x\mathrm{d}x + \frac{1}{2}y_0\mathrm{e}^x\mathrm{d}x\mathrm{d}x,$$

其中 $\mathrm{d}x = (r - \frac{1}{2}\sigma^2)\mathrm{d}t + \sigma\mathrm{d}B_t$. 利用 Ito 公式中的微分恒等式, 得到

$$\mathrm{d}x\mathrm{d}x = \sigma^2\mathrm{d}t.$$

所以

$$\begin{aligned}\mathrm{d}y &= y_0\mathrm{e}^x\left(r - \frac{1}{2}\sigma^2\right)\mathrm{d}t + y_0\mathrm{e}^x\sigma\mathrm{d}B_t + \frac{1}{2}y_0\sigma^2\mathrm{e}^x\mathrm{d}t \\ &= y_0\mathrm{e}^x r\mathrm{d}t + y_0\mathrm{e}^x\sigma\mathrm{d}B_t = ry\mathrm{d}t + \sigma y\mathrm{d}B_t.\end{aligned}$$

◀

图 9-12 给出了具有常数**漂移系数** (drift coefficient) r 和**扩散系数** (diffusion coefficient) σ 的几何布朗运动的实现. 该模型广泛应用于金融建模中. 特别地, 几何布朗运动是 Black-Scholes 方程的基础模型, 后者用于金融衍生产品的定价.

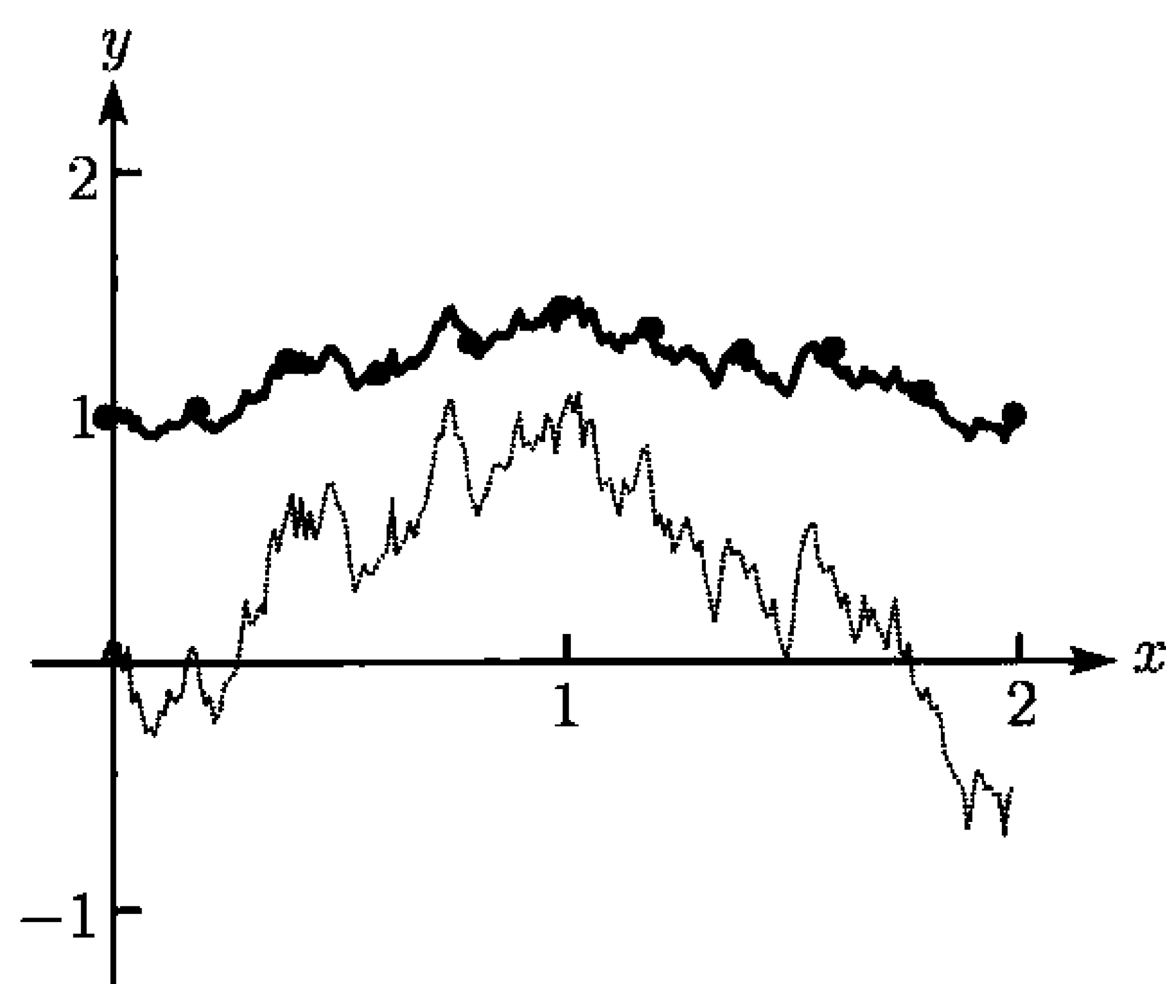


图 9-12 指数型布朗运动 SDE (9.19) 的解. (9.18) 的准确解由图中实曲线表示, 其 Euler-Maruyama 近似由圆点表示. 点曲线表示的是相应实现的布朗运动路线. 参数为 $r = 0.1$, $\sigma = 0.3$, $\Delta t = 0.2$

例 9.11 和例 9.12 是特例. 就像 ODEs 中的情形, 相当有限的 SDEs 具有闭形式的解. 更普遍地, 常常需要用数值逼近的办法来求解.

9.4.2 随机微分方程的数值方法

我们可以用和第 6 章 Euler 方法类似的方法去求随机微分方程的近似解. 就如 Euler 所做的工作那样, Euler-Maruyama 方法也是依靠离散时间轴来实现. 我们

在下面的网格点上定义近似解:

$$a = t_0 < t_1 < t_2 < \cdots < t_n = b.$$

并且分别对于不同的 t 点, 定义不同的 y 值为

$$w_0 < w_1 < w_2 < \cdots < w_n.$$

给定一个随机微分方程初值问题

$$\begin{cases} \mathrm{d}y(t) = f(t,y)\mathrm{d}t + g(t,y)\mathrm{d}B_t, \\ y(a) = y_a \end{cases} \tag{9.20}$$

可以近似地计算出它的解:

Euler-Maruyama 方法

$w_0 = y_0$

for $i = 0, 1, 2, \cdots$

$$w_{i+1} = w_i + f(t_i, w_i)(\Delta t_i) + g(t_i, w_i)(\Delta B_i) \tag{9.21}$$

end

$\vdots$

其中

$$\Delta t_i = t_{i+1} - t_i \tag{9.22}$$

$$\Delta B_i = B_{t_{i+1}} - B_{t_i}$$

关键之处在于如何去建立布朗运动 ΔB_i 的模型. 定义 $N(0,1)$ 为标准随机变量的集合, 它们服从均值为 0、标准差为 1 的正态分布. 每个随机数 ΔB_i 由 9.3.2 节介绍的方法来计算, 即

$$\Delta B_i = z_i\sqrt{\Delta t_i}, \tag{9.23}$$

其中 z_i 是 $N(0,1)$ 中的一个随机变量. 在 MATLAB 中, z_i 可由 `randn` 命令来生成. 再次注意到与确定性常微分方程的不同之处. 我们计算得到的每一个集合 $\{w_0, \cdots, w_n\}$ 都是随机过程 $y(t)$ 的近似解实现, 它依赖于随机数 Z_i 的选取. 因为 B_t 是一个随机过程, 每次实现肯定是不一样的, 所以我们得到的近似解也肯定是不一样的.

作为第一个例子, 我们展示如何将 Euler-Maruyama 方法应用于指数型布朗运动随机微分方程 (9.19). 根据 (9.21), Euler-Maruyama 方法有如下形式:

$$\begin{aligned} w_0 &= y_0 \\ w_{i+1} &= w_i + rw_i(\Delta t_i) + \sigma w_i(\Delta B_i). \end{aligned} \tag{9.24}$$

准确解 [由解 (9.18) 生成] 及其相应的 Euler-Maruyama 近似如图 9-12 所示. "相应的", 意思是该近似方法使用了与正确解相同的布朗运动实现 (也在图 9-12 中标出) 的近似. 留意一下准确解曲线和近似解点的紧密一致性. 近似解点由每隔 0.2 个时间单位的小圆点表示.

例 9.13 用数值方法求解 **Langevin 方程**

$$dy = -ry\mathrm{d}t + \sigma \mathrm{d}B_t, \tag{9.25}$$

其中 r 和 σ 是正的常数.

和前面的例子不同, 这里不可能像普通情形那样对方程进行求导. Langevin 方程的解是一个随机过程, 称为 **Ornstein-Uhlenbeck 过程**. 图 9-13 表示出了它的一个近似解. 近似解是由一个 Euler-Maruyama 近似生成, 其中采用了步长

$$\begin{aligned} w_0 &= y_0 \\ w_{i+1} &= w_i - rw(\Delta t_i) + \sigma(\Delta B_i), \quad i = 1, \cdots, n \end{aligned} \tag{9.26}$$

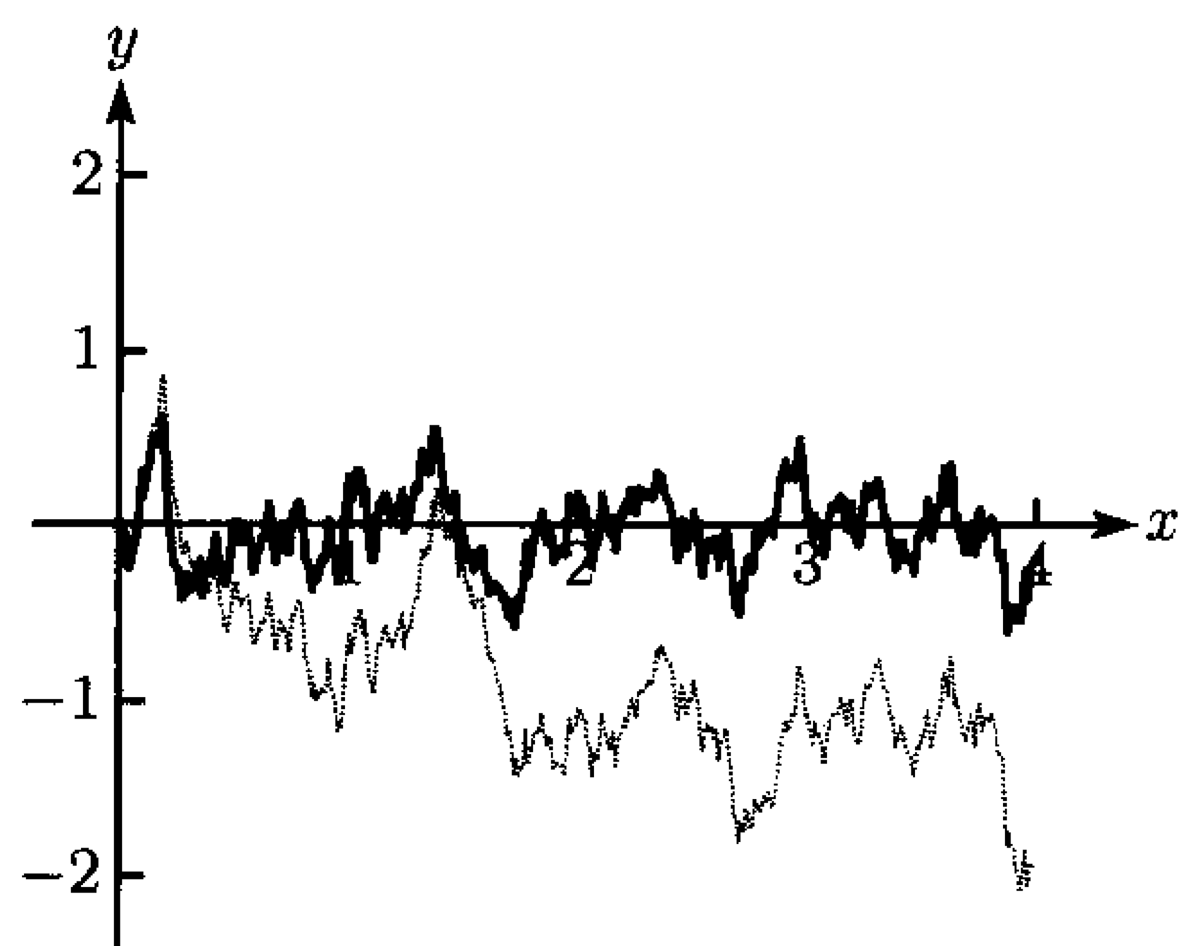


图 9-13 Langevin 方程 (9.25) 的解. 上面的曲线是由 Euler-Maruyama 方法计算得到的近似解, 其中参数 $r = 10, \sigma = 1$. 点曲线是相应的布朗运动实现

在有噪声存在的背景下, 即将回到某个状态 (本例中即 $y = 0$ 的状态) 的系统常用这个随机微分方程来建模. 我们可以把它想象成一个盛满乒乓球的碗放在车里, 车子在崎岖的路面上行走. 乒乓球从碗中心偏离的距离 $y(t)$ 可以由 Langevin 方程来建模. ◀

接下来讨论随机微分方程解的 "阶" 概念. 随机微分方程的解是一个随机过程, 每个计算出来的轨道只是过程的一个实现而已. 除此之外, 它和一般常微分方程的解是类似的. 每个布朗运动的实现将会强制解 $y(t)$ 的一个不同的实现. 如果我们在 t 轴上固定一个点 $T > 0$, 每个从 $t = 0$ 开始取值的解将会在 T 点返回一个随机数. 也就是说, $y(T)$ 是一个随机变量. 而且, 每个 (比如说) 由 Euler-Maruyama 方法计算出来的解路径 $w(t)$, 在 T 点也将返回一个随机数. 因此 $w(T)$ 也是一个随机

变量. 在时间 T, 两个变量的差 $e(T)=y(T)-w(T)$ 也是一个随机变量. 就像常微分方程解的分析方法一样, "阶" 的概念量化了误差 $e(T)$ 的期望值.

定义 9.3 如果误差的期望值是步长的 m 次方阶, 则说一个常微分方程解的**阶** (order) 是 m. 也就是说, 当步长 Δt 趋向于 0 时, 对于任意的时间 T, $\mathrm{E}\{|y(T)-w(T)|\}=O((\Delta t)^m)$.

令人惊讶的是, 不像常微分方程情形那样, Euler 方法是 1 阶的, 随机微分方程的 Euler-Maruyama 方法是 1/2 阶的. 为了建立随机微分方程的 1 阶方法, 我们需要把 "随机泰勒级数" 的概念引入方法.

考虑随机微分方程

$$\begin{cases} \mathrm{d}y(t)=f(t,y)\mathrm{d}t+g(t,y)\mathrm{d}B_t \\ y(0)=y_0 \end{cases}$$

Milstein 方法

$w_0=y_0$

for $i=0,1,2,\cdots$

$$w_{i+1}=w_i+f(t_i,w_i)(\Delta t_i)+g(t_i,w_i)(\Delta B_i)+\frac{1}{2}g(t_i,w_i)\frac{\partial g}{\partial y}(t_i,w_i)((\Delta B_i)^2-\Delta t_i)$$

end (9.27)

Milstein 方法是 1 阶的. 注意到, 如果方程里的扩散部分 $g(y,t)$ 没有 y 项的话, Milstein 方法等价于 Euler-Maruyama 方法. 当步长 h 趋于 0 的时候, Milstein 方法会比 Euler-Maruyama 方法更快地趋向于准确的随机过程解.

例 9.14 将 Milstein 方法用于几何布朗运动.

方程为

$$\mathrm{d}y=ry\mathrm{d}t+\sigma y\mathrm{d}B_t, \tag{9.28}$$

其解为

$$y=y_0\mathrm{e}^{(r-\frac{1}{2}\sigma^2)t+\sigma B_t}. \tag{9.29}$$

前面已经讨论了它的 Euler-Maruyama 近似. 使用固定的步长 Δt, Milstein 方法就变成

$$\begin{aligned} w_0&=y_0, \\ w_{i+1}&=w_i+rw_i(\Delta t)+\sigma w_i(\Delta B_t)+\frac{1}{2}\sigma^2 w_i((\Delta B_t)^2-\Delta t). \end{aligned} \tag{9.30}$$

对 Euler-Maruyama 方法和 Milstein 方法使用逐步减小的步长 Δt, 将会得到近似效果的成功改进, 如表 9-6 所示.

表中的两列数据表示误差 $|w(T)-y(T)|$ 在 $T=8$ 时的平均值 (对 100 个实现求平均). 注意到 $w(t)$ 和 $y(t)$ 有同样的布朗运动增量 ΔB_i. 在表中可清楚地看到

Euler-Maruyama 方法是 1/2 阶的, Milstein 方法是 1 阶的, 为了使 Euler-Maruyama 方法的误差减小到原来的 $\frac{1}{2}$, 我们必须把步长除以 4. 对于 Milstein 方法而言, 把步长除以 2 就得到相同的结果. 表中的数据取对数单位后的图见图 9-14. ◀

表　9-6

Δt	Euler-Maruyama	Milstein
2^{-1}	0.169 369	0.063 864
2^{-2}	0.136 665	0.035 890
2^{-3}	0.086 185	0.017 960
2^{-4}	0.060 615	0.008 360
2^{-5}	0.048 823	0.004 158
2^{-6}	0.035 690	0.002 058
2^{-7}	0.024 277	0.000 981
2^{-8}	0.016 399	0.000 471
2^{-9}	0.011 897	0.000 242
2^{-10}	0.007 913	0.000 122

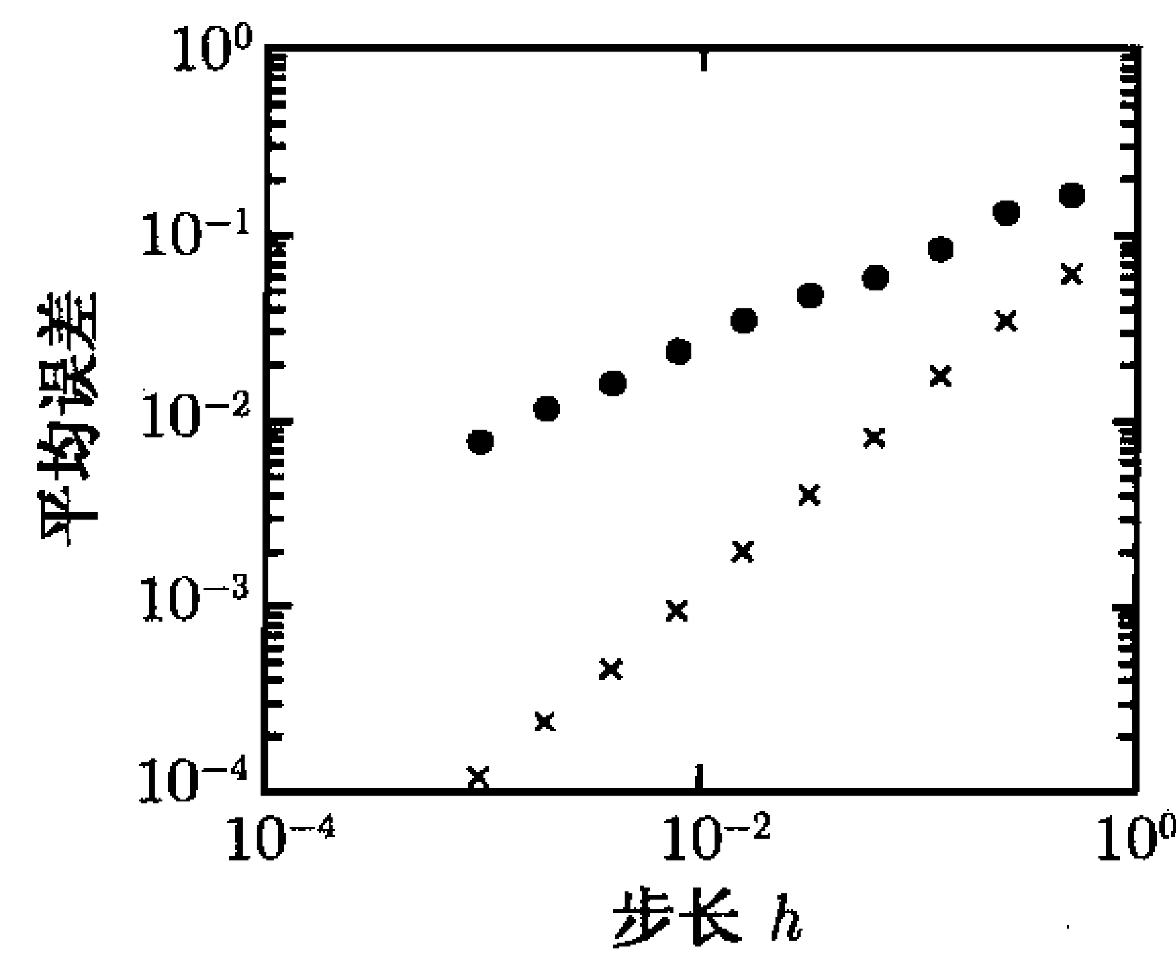


图 9-14　Euler-Maruyama 方法和 Milstein 方法的误差. 几何布朗运动方程 (9.28) 的解路径, 并和 (9.29) 给出的准确解作比较. 两种方法误差的绝对值与步长 h 的关系在图中画出. Euler-Maruyama 方法的误差由圆点表示, Milstein 方法的误差由叉号表示. 注意到在双对数图中, 两个斜率分别是 1/2 和 1

亮点　收敛性

这里引入随机微分方程数值方法的阶, Euler-Maruyama 方法的阶是 1/2, Milstein 方法的阶是 1. 如果以常微分方程的标准来看, 都属于低阶方法. 随机微分方程的高阶方法当然也可以构造出来, 但是当阶增大的时候, 方法的复杂性会大大增加. 在实际应用中, 是否需要高阶方法取决于如何使用最后得到的近似解. 在常微分方程的情形, 一般的假定是: 初始条件和微分方程必须是高精度的. 这样, 尽可能以相同的精度计算它的解

才有意义, 这时如果高阶方法的代价不高, 就使用高阶的方法. 但是在很多情形下, SDE 的高阶方法的优势并不明显, 如果这些高阶方法伴随着额外的计算耗费, 那么这些解法也可能被认为是不正确的.

例 9.15 用 Euler-Maruyama 方法和 Milstein 方法求解随机微分方程

$$dy = -2e^{-2y}dt + 2e^{-y}dB_t. \tag{9.31}$$

这个例子有一个有趣而值得注意的性质值得讨论. 可以找到一个显式的解, 但是这个解只在有限的时间扩张里存在. 用 Ito 公式 (9.17), 只要对数符号里面的量是正的, $y(t) = \ln(2B_t + e^{y_0})$ 就是一个解. 在当布朗运动实现使得 $2B_t + e^{y_0}$ 变成负数的那个第一时间 t, 以及 t 以后的时刻, 解便不复存在.

对于这个方程, Euler-Maruyama 方法是

$$\begin{aligned} &w_0 = y_0, \\ &w_{i+1} = w_i - 2e^{-2w_i}(\Delta t_i) + 2e^{-w_i}(\Delta B_i), \end{aligned} \tag{9.32}$$

Milstein 方法是

$$\begin{aligned} &w_0 = y_0, \\ &w_{i+1} = w_i - 2e^{-2w_i}(\Delta t_i) + 2e^{-w_i}(\Delta B_i) - 2e^{-2w_i}((\Delta B_i)^2 - \Delta t_i). \end{aligned} \tag{9.33}$$

在区间 $0 \leqslant t \leqslant 3$ 上的解如图 9-15 所示. ◀

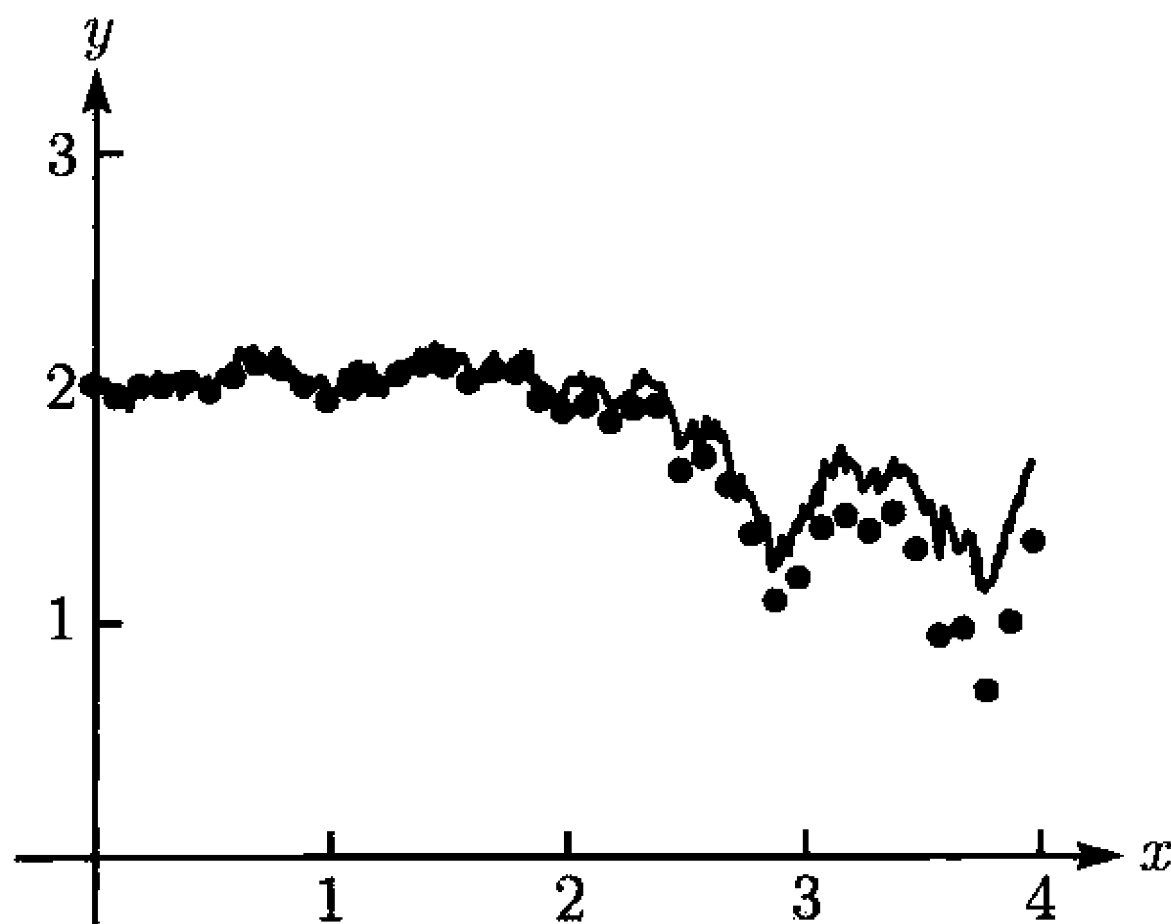


图 9-15 方程 (9.31) 的解

准确解和用圆点表示的 Milstein 近似解. 到现在为止, 我们见到的随机过程都有着随 t 变大而增大的方差. 例如布朗运动的方差是 $\mathrm{Var}(B_t) = t$. 我们用一个值得注意的例子来结束本节. 其实现的末端就如始端一样可以预测

例 9.16 **数值求解布朗桥随机微分方程**

$$\begin{cases} dy = \dfrac{y_1 - y}{t_1 - t}dt + dB_t, \\ y(t_0) = y_0, \end{cases} \tag{9.34}$$

其中 y_1 和 $t_1 > t_0$ 是给定的数.

布朗桥 (9.34) 的解如图 9-16 所示. 当创建路径的时候, 由于目标斜率适当地改变, 所有解过程的实现都终于一个我们期望的点 (t_1, y_1). 这个解路径就可以看成是两个给定的点 (t_0, y_0) 和 (t_1, y_1) 之间的随机生成的 "桥". ◀

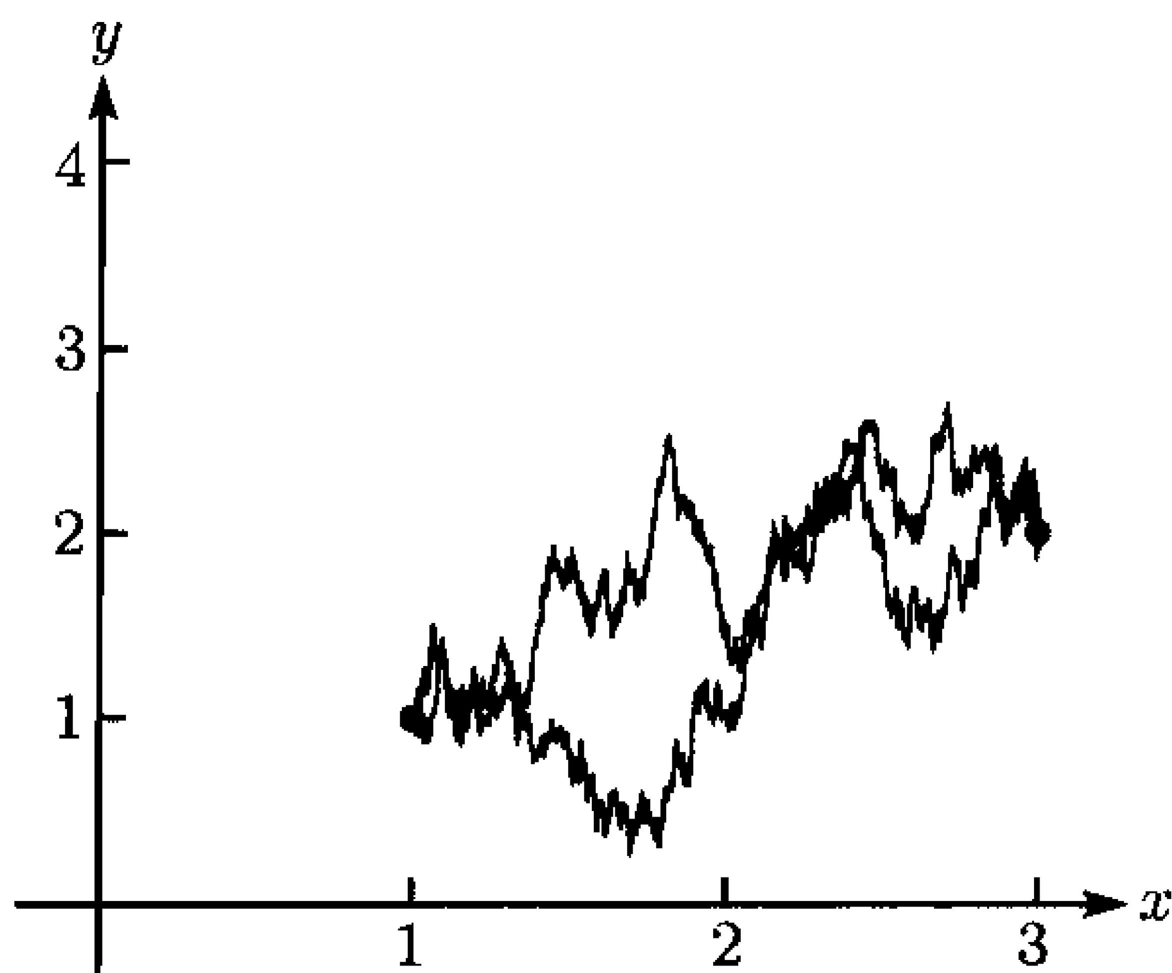


图 9-16　布朗桥. 方程 (9.34) 的解的两个实现. 端点是 $(t_0, y_0) = (1, 1)$ 和 $(t_1, y_1) = (3, 2)$

习题 9.4

1. 用 Ito 公式证明 SDE 初值问题

$$\text{(a)} \begin{cases} dy = B_t dt + t dB_t, \\ y(0) = c, \end{cases} \quad \text{(b)} \begin{cases} dy = 2B_t dB_t, \\ y(0) = c \end{cases}$$

的解分别是 (a) $y(t) = tB_t + c$; (b) $y(t) = B_t^2 - t + c$.

2. 用 Ito 公式证明 SDE 初值问题

$$\text{(a)} \begin{cases} dy = (1 - B_t^2)e^{-2y} dt + 2B_t e^{-y} dB_t, \\ y(0) = 0, \end{cases} \quad \text{(b)} \begin{cases} dy = B_t dt + \sqrt[3]{9y^2} dB_t, \\ y(0) = 0 \end{cases}$$

的解分别是 (a) $y(t) = \ln(1 + B_t^2)$,　(b) $y(t) = \frac{1}{3}B_t^3$.

3. 用 Ito 公式证明 SDE 初值问题

$$\text{(a)} \begin{cases} dy = ty dt + e^{t^2/2} dB_t, \\ y(0) = 1, \end{cases} \quad \text{(b)} \begin{cases} dy = 3(B_t^2 - t) dB_t, \\ y(0) = 0 \end{cases}$$

的解分别是 (a) $y(t) = (1 + B_t)e^{t^2/2}$,　(b) $y(t) = B_t^3 - 3tB_t$.

4. 用 Ito 公式证明 SDE 初值问题

$$\text{(a)} \begin{cases} dy = -\frac{1}{2} y dt + \sqrt{1 - y^2} dB_t, \\ y(0) = 0, \end{cases} \quad \text{(b)} \begin{cases} dy = y(1 + 2\ln y) dt + 2yB_t dB_t, \\ y(0) = 1 \end{cases}$$

的解分别是 (a) $y(t) = \sin B_t$;　(b) $y(t) = e^{B_t^2}$.

5. 用 Ito 公式证明方程 (9.31) 的解是 $\ln(2B_t + e^{y_0})$.

6. (a) 求解拟布朗桥常微分方程问题

$$\begin{cases} y' = \frac{y_1 - y}{t_1 - t}, \\ y(t_0) = y_0. \end{cases} \tag{9.35}$$

这些解像布朗桥问题那样过点 (t_1, y_1) 吗? 对下面的方程回答相同的问题:

(b) $\begin{cases} y' = \frac{y_1 - y_0}{t_1 - t_0}, \\ y(t_0) = y_0; \end{cases}$ (c) $\begin{cases} \mathrm{d}y = \frac{y_1 - y_0}{t_1 - t_0}\mathrm{d}t + \mathrm{d}B_t, \\ y(t_0) = y_0. \end{cases}$

计算机问题 9.4

1. 用 Euler-Maruyama 方法求习题 1 中 SDE 初值问题的近似解. 使用初值 $y(0) = 0$. 画出准确解的图像 (用相同的随机增量, 追踪布朗运动 B_t 而得到), 并且画出在区间 $[0, 10]$ 的近似解的图像, 步长取 $h = 0.01$. 画出在给定区间上误差的半对数图像.
2. 用 Euler-Maruyama 方法求习题 2 中 SDE 初值问题的近似解. 初值 $y(0) = 1$. 画出准确解的图像, 并且画出在区间 $[0, 1]$ 的近似解的图像, 步长取 $h = 0.01$. 画出在给定区间上误差的半对数图像.
3. 在区间 $[0, 2]$ 上, 步长取 $h = 0.01$, 用 Euler-Maruyama 方法求解习题 3. 画出随机过程解的两个实现.
4. 在区间 $[0, 1]$ 上, 步长取 $h = 0.01$, 用 Euler-Maruyama 方法求解习题 4. 画出随机过程解的两个实现.
5. 对于区间 $[0, 1]$ 上的问题

$$\begin{cases} \mathrm{d}y = B_t \mathrm{d}t + \sqrt[3]{9y^2}\mathrm{d}B_t, \\ y(0) = 0, \end{cases}$$

步长分别取 $h = 0.1, 0.01, 0.001$, 用 Euler-Maruyama 方法求解. 对于每一种步长, 运行 5 000 个近似解的实现, 并在 $t = 1$ 时求平均误差. 对于不同的步长, 把 $t = 1$ 时的平均误差列表显示出来. 这些平均误差的阶和理论一致吗?

6. 用 Euler-Maruyama 方法求解 SDE 初值问题 $\mathrm{d}y = y\mathrm{d}t + y\mathrm{d}B_t, y(0) = 1$. 画出近似解与准确解 $y(t) = \mathrm{e}^{\frac{1}{2}t + B_t}$ 的图像. 步长 $h = 0.1$, 区间为 $0 \leqslant t \leqslant 2$.
7. 用 Milstein 方法求习题 2b 的 SDE 初值问题的近似解. 在区间 $[0, 5]$ 上画出准确解和近似解的图像, 步长 $h = 0.1$. 在区间上画出误差的半对数图像.
8. 用 Milstein 方法求习题 4a 的 SDE 初值问题的近似解. 在区间 $[0, 5]$ 上画出准确解和近似解的图像, 步长 $h = 0.1$. 在区间上画出误差的半对数图像.
9. 对于区间 $[0, 1]$ 上的问题

$$\begin{cases} \mathrm{d}y = B_t \mathrm{d}t + \sqrt[3]{9y^2}\mathrm{d}B_t, \\ y(0) = 0, \end{cases}$$

步长分别取 $h = 0.1, 0.01, 0.001$, 用 Milstein 方法求近似解. 对于每一个步长, 运行 5 000 个近似解的实现, 并在 $t = 1$ 处求平均误差. 对于不同的步长, 把 $t = 1$ 时的平均误差列表显示出来. 这些平均误差的阶和理论一致吗?

10. $y(t)$ 是 Langevin 方程

$$\begin{cases} \mathrm{d}y = -y\mathrm{d}t + \mathrm{d}B_t, \\ y(0) = \mathrm{e} \end{cases}$$

的 Euler-Maruyama 解, 对 $y(1)$ 进行蒙特卡罗估计. 要求平均 $n = 1\ 000$ 个实现, 其中步长 $h = 0.01$. 和 $y(1)$ 的期望值 1 作一下比较.

实例检验 9 Black-Scholes 公式

蒙特卡罗模拟和随机微分方程模型大量应用于金融计算中. **金融衍生产品** (financial derivative) 是一种其价值是衍生自其他工具价值的金融工具. 特别地, **期权** (option) 是完成某个特定金融交易的权利, 但不是义务.

(欧式) 看涨期权 (call option) 是在未来的一个日期 [称为**执行日** (exercise date)] 里购买以一个预定价格 [称为**交割价** (strike price)] 购买记券股份的权利. 购买权买进或者卖出, 对于公司来说是为了管理风险; 而对于个人和公共基金, 是投资策略的一部分. 我们的目标是计算看涨期权的价值.

例如, ABC 公司 12 月份的看涨期权 15 美元, 表示在 12 月份以每股 15 美元的价格购买的权利. 假设在 6 月 1 号的价格是 12 美元. 这样一项权利的价值是多少? 在执行日那天, K 美元的看涨期权是确定的: $\max(X-K,0)$ 就是看涨期权的价值, 其中 X 是股票市场的当前价格. 那是因为, 如果 $X>K$, 以 K 美元的价格购买 ABC 的权利值 $(X-K)$ 美元; 如果 $X<K$, 以 K 美元购买 ABC 的权利变得一文不值, 因为我们可以用更低的价格买多少都可以. 虽然在执行日那天, 期权的价值是清楚的, 困难之处在于截止日期前的一段时间如何计算期权的价值.

在 20 世纪 60 年代, Fisher Black 和 Myron Scholes 探讨了关于几何布朗运动的假设

$$\mathrm{d}X = mX\mathrm{d}t + \sigma X\mathrm{d}B_t, \tag{9.36}$$

并以此作为股票市场的模型, 其中 m 是平移, 或者叫股票市场的增长率, σ 是扩散常数或者叫**波动** (volatility). m 和 σ 都可以由过去股票价格数据估计出来. Black 和 Scholes 的过人之处在于发展了套汇理论. 这个理论是在现行利率 r 下, 通过明智地平衡股票持有和现金借贷来复制期权. 他们断言的结果是: 正确的看涨期权价值 (还有 T 年到期) 是当前期望的期权价值在到期日的现值. 标的股票价格 (underlying stock price) $X(t)$ 满足 SDE

$$\mathrm{d}X = rX\mathrm{d}t + \sigma X\mathrm{d}B_t, \tag{9.37}$$

也就是说, 对于 $t=0$ 时刻的股票价格 X_0, 截止期限是 $t=T$ 的期权价值是下面式子的期望值:

$$C(X_0,T) = \mathrm{e}^{-rT}E(\max(X(T)-K,0)), \tag{9.38}$$

其中, $X(t)$ 由 (9.37) 给出. 在它们的衍生物中, 令人吃惊的是, 在 (9.37) 中用利率 r 代替了 (9.36) 的平移 m. 事实上, 预计中股票的增长率竟然和期权价格无关! 这与 Black-Scholes 理论的基石无套汇假设相一致. 无套汇假设是说在有效的市场里, 不存在无风险收入.

(9.38) 依赖于随机变量 $X(T)$ 的期望值, 这是在模拟过程中唯一可用的随机变量. 于是, 除了这种观点之外, Black 和 Scholes[1] 对于看涨期权价格提供了一个封闭形式的表达式, 那就是

$$C(X,T) = XN(d_1) - K\mathrm{e}^{-rT}N(d_2), \tag{9.39}$$

其中 $N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \mathrm{e}^{-s^2/2} \mathrm{d}s$ 是正态累计分布函数,

$$d_1 = \frac{\ln(X/K) + \left(r + \frac{1}{2}\sigma^2\right) T}{\sigma\sqrt{T}}, \quad d_2 = \frac{\ln(X/K) + \left(r - \frac{1}{2}\sigma^2\right) T}{\sigma\sqrt{T}}$$

式 (9.39) 称为 **Black-Scholes 公式**.

建议习题

假设 ABC 公司的股票是每股 12 美元. 考虑欧式看涨期权, 期权交割价是 15 美元, 执行日是从今天开始的 6 个月, 因此 $T = 0.5$ 年. 假设有一个固定的利率 $r = 0.05$, 股票的波动是 0.25 (也就是说每年 25%).

1. 用蒙特卡罗模拟计算 (9.38) 式的期望值. 用 Euler-Maruyama 方法求 (9.37) 的近似解, 取步长 $h = 0.01$, 初始值 $X_0 = 12$. 注意到 SDE (9.36) 与计算不相关. 至少重复执行 10 000 次.

2. 由 Black-Scholes 公式 (9.39), 比较你第 1 步的近似值和准确值. 函数 $N(x)$ 可以用 MATLAB 误差函数 **erf** 来计算:

$$N(x) = (1 + \mathrm{erf}(x/\sqrt{2}))/2$$

3. 用 Milstein 方法代替 Euler-Maruyama 方法, 并且重复第 1 步、对两种方法的误差进行比较.

4. (欧式) 看跌期权 (put option) 和看涨期权不一样, 因为它表示的是以交割价来售出而不是买入的权利. 看跌期权的价值是

$$P(X,T) = \mathrm{e}^{-rT} E(\max(K - X(T), 0)), \tag{9.40}$$

使用 (9.37) 中提供的 $X(T)$. 对于第一步相同的数据, 通过蒙特卡罗模拟计算这个价值, 同时采用 Euler-Maruyama 方法和 Milstein 方法.

5. 比较第 4 步的近似结果与用 Black-Scholes 公式计算的看跌期权

$$P(X,T) = K\mathrm{e}^{-rT} N(-d_2) - XN(-d_1) \tag{9.41}$$

6. 障碍期权 (barrier option) 有一种支付 (payout), 如果股票超过一个给定的水平, 这种支付就会取消掉. 考虑障碍期权, 交割价 $K = 15$ 美元, 障碍 $L = 10$ 美元. 如果对于 $0 < t < T, X(t) > L$, 则支付是 $\max(X - K, 0)$; 否则支付就是 0. 设计并执行一个蒙特卡罗模拟, 使用几何布朗运动 (9.36) 和 (9.38) 并为障碍期权支付进行修改. 与准确值

$$V(X,T) = C(X,T) - \left(\frac{X}{L}\right)^{1-2r/\sigma^2} C(L^2/X, T)$$

进行比较, 其中, $C(X,T)$ 是标准欧式看涨期权价值, 交割价是 K. 参考 [21, 8] 可获得更多关于奇异期权、奇异期权的价格公式, 以及蒙特卡罗模拟在经济中的作用等方面的细节.

软件及进一步的阅读

教科书 [4] 简述了随机数生成问题. 这个领域中其他的经典资源是 [12, 16]. "随机数生成方法" 与 "一般估计准则讨论" 的比较参见 [6].

randu 问题在 [13] 中叙述. 最小标准生成元在 [18] 中介绍. MATLAB 的随机数生成元是基于 Marsaglia 和 Zaman 在 [14] 中介绍的 "借位减" 方法而成. 有关蒙特卡罗方法及其应用的更多知识在 [3, 19] 可找到.

随机微分方程方面的现代教科书包含 [17, 9, 20]. 在这个领域上做研究, 需要基础概率论方面扎实的功底. 关于随机微分方程数值解法方面的内容, [10] 里有广泛的论述. [11] 是一本更面向应用的手册. 文章 [7] 是 MATLAB 关于随机微分方程基本解法软件的介绍, 非常浅显易懂. [20] 简要介绍了随机微分方程在金融方面的大量应用.

第10章　三角插值和快速 Fourier 变换

数字信号处理 (DSP) 芯片是高级消费电子产品中的核心. 移动电话、CD 和 DVD 控制器、汽车电子器件、个人数字助理、数字调制解调器、照相机以及电视机都使用了这些无所不在的芯片. 数字信号处理芯片的特点是它的快速数值计算的能力, 包括快速 Fourier 变换 (FFT).

DSP 的最基本功能之一是通过滤波掉不想要的噪声以分离出想要的输入信息. 创建可靠的语言识别软件是目前正在进行的研究, 其中重要的部分是从杂乱的背景中提取信号. 它也是参加足球比赛的机器狗所使用的模式识别装置的关键因素, 把感知的输入转成可用的数据资料.

实例检验　10.3.3 节后的实例检验 10 叙述了 Wiener 滤波器, 一种借助于 DSP 来减小噪声的基本构造块.

半个世纪以前, 即使是最乐观的三角学教师, 也不可能预见到正弦和余弦对现代技术的影响. 如我们在第 4 章中学过的, 多频率的三角函数是周期数据自然的插值函数. Fourier 变换在执行插值中相当有效, 并且在现代信号处理的数据密集型应用中是不可代替的.

三角插值的效能与正交性概念有密切关系. 我们将看到正交基函数使得插值和数据的最小二乘拟合更加简单而且更加精确. Fourier 变换用了这种正交性并且提供了用正弦和余弦插值的有效方法. Cooley 和 Tukey 在计算上的突破被称为快速 Fourier 变换（FFT）, 意味着可以非常廉价地计算离散 Fourier 变换（DFT）.

第 10 章覆盖了离散 Fourier 变换的基本概念, 包括对复数的简短介绍. 在三角插值和最小二乘逼近中, DFT 的作用是有特色的, 而且被看作用正交基函数进行逼近的一种特殊情形. 这是数字滤波和信号处理的本质.

10.1　Fourier 变换

法国数学家 Jean Baptiste Joseph Fourier, 在法兰西大革命时期从铡刀下逃生, 又在拿破仑麾下参加战争, 他抽时间研究了热传导理论. 为了使这种理论起作用, 他需要扩张函数, 不是像 Taylor 级数那样用多项式来表示, 而是用首先由 Euler 和 Bernoulli 建立的一种革命性的方法——用正弦和余弦函数来表示. 虽然当时被认为不够严谨而受到主流数学家的反对, 但今天 Fourier 方法已渗透到应用数学、物

理和工程的许多领域. 本节引入离散 Fourier 变换, 并描述一种有效的算法来计算它, 即快速 Fourier 变换.

10.1.1　复算术

通过采用复数的语言可以大大简化三角函数的记法要求. 每个复数具有形式 $z = a + b\mathrm{i}$, 这里 $\mathrm{i} = \sqrt{-1}$. 如图 10-1 所示, 每一个 z 在几何上表示为沿实 (水平) 轴长度为 a, 沿虚 (垂直) 轴长度为 b 的二维向量. 数 $z = a + b\mathrm{i}$ 的**复度量** (complex magnitude) 定义为 $|z| = \sqrt{a^2 + b^2}$, 恰好等于在复平面上从原点到这个复数的距离. 复数 $z = a + b\mathrm{i}$ 的**复共轭** (complex conjugate) 是 $\bar{z} = a - b\mathrm{i}$.

复算术中著名的 **Euler 公式**是指 $\mathrm{e}^{\mathrm{i}\theta} = \cos\theta + \mathrm{i}\sin\theta$.$z = \mathrm{e}^{\mathrm{i}\theta}$ 的复度量是 1, 因此, 如图 10-2 所示, 这种形式的复数落在复平面的单位圆上. 可以把任意复数 $a + b\mathrm{i}$ 表示成**极坐标** (polar representation)

$$z = a + b\mathrm{i} = r\mathrm{e}^{\mathrm{i}\theta}, \tag{10.1}$$

这里 r 的复度量是 $|z| = \sqrt{a^2 + b^2}, \theta = \arctan\frac{b}{a}$.

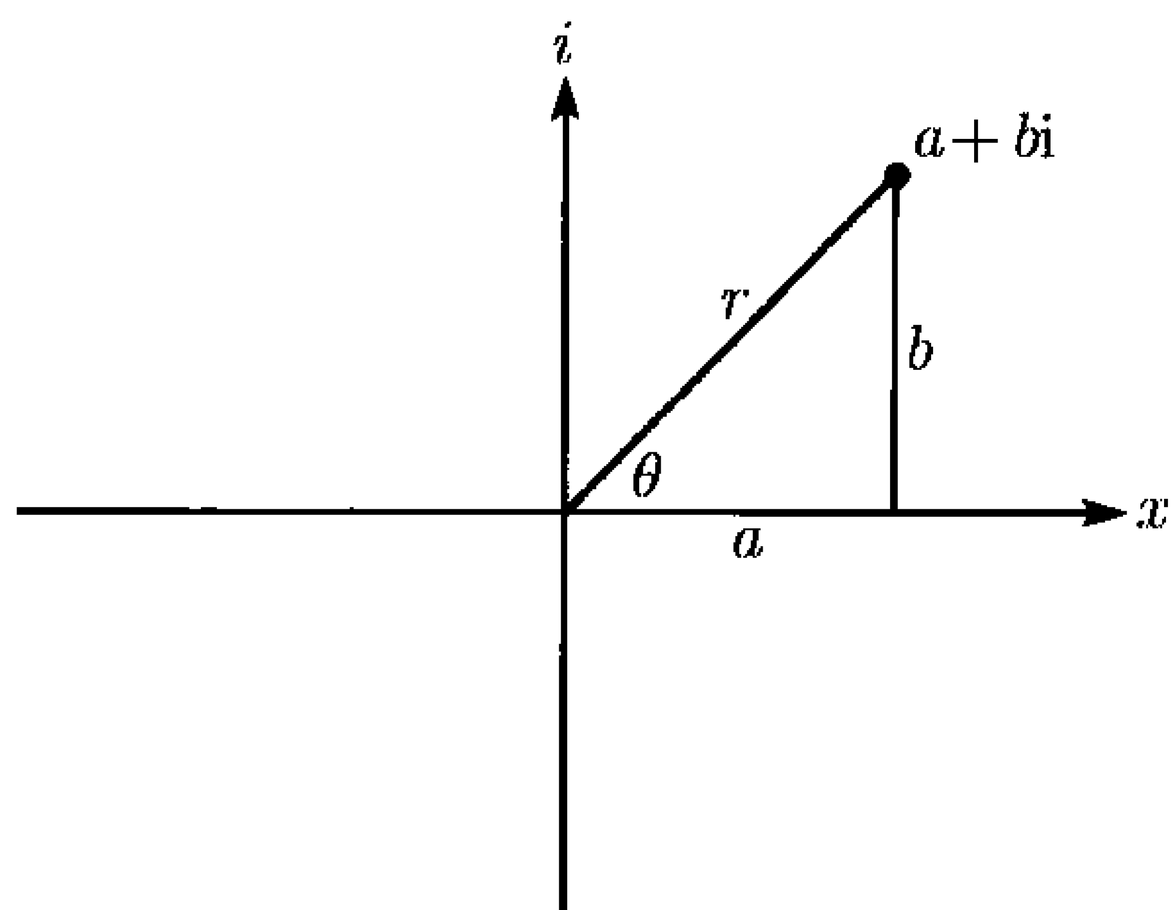


图 10-1　复数的表示: 实部与虚部分别为 a 和 $b\mathrm{i}$. 极坐标表示是 $a + b\mathrm{i} = r\mathrm{e}^{\mathrm{i}\theta}$

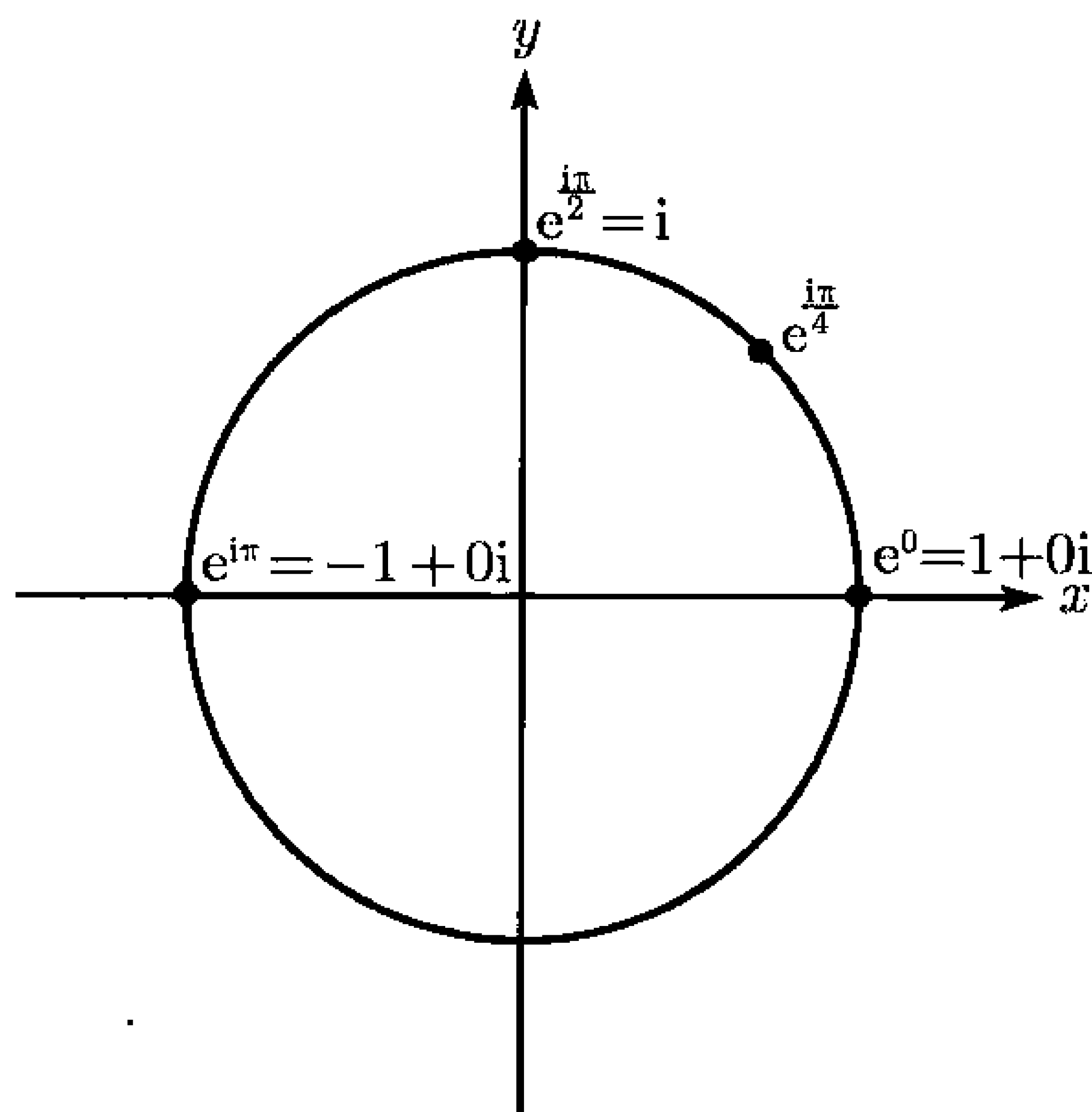


图 10-2　复平面中的单位圆. 关于某个角度 θ 的形如 $\mathrm{e}^{\mathrm{i}\theta}$ 的复数有度量 1 并且落在单位圆上

复平面中的单位圆与度量 $r = 1$ 的复数相对应. 为了把单位圆的两个数 $\mathrm{e}^{\mathrm{i}\theta}$ 和 $\mathrm{e}^{\mathrm{i}\gamma}$ 相乘, 我们可以先转换成三角函数, 然后再相乘:

$$\begin{aligned}\mathrm{e}^{\mathrm{i}\theta}\mathrm{e}^{\mathrm{i}\gamma} &= (\cos\theta + \mathrm{i}\sin\theta)(\cos\gamma + \mathrm{i}\sin\gamma)\\ &= \cos\theta\cos\gamma - \sin\theta\sin\gamma + \mathrm{i}(\sin\theta\cos\gamma + \sin\gamma\cos\theta).\end{aligned}$$

回忆余弦加法公式和正弦加法公式, 可以把它写成

$$\cos(\theta + \gamma) + \mathrm{i}\sin(\theta + \gamma) = \mathrm{e}^{\mathrm{i}(\theta+\gamma)}.$$

等价地, 恰好是指数相加

$$e^{i\theta}e^{i\gamma} = e^{i(\theta+\gamma)}. \tag{10.2}$$

式 (10.2) 表示：单位圆上两个数的乘积给出了单位圆上一个新的点，它的角度是这两个数的角度之和. Euler 公式隐藏了像正弦和余弦加法公式这样的三角学细节, 而且使记法更简单. 这就是我们把复算术引入到三角插值中的原因. 尽管它完全可以在实数内完成, 但是 Euler 公式有深刻的简化效果.

我们挑选出度量为 1 的复数的一类特殊子集. 如果 $z^n = 1$, 那么复数 z 是 n 次**单位根**. 实数轴上仅有两个单位根, -1 和 1; 然而在复平面上却有许多个. 例如, 因为 $i^4 = (-1)^2 = 1$, 所以 i 本身是 4 次单位根.

对任意的 $k < n$, 如果 n 次单位根不是 k 次单位根, 那么这个 n 次单位根称为**本原的** (primitive). 根据这个定义, -1 是一个本原二次单位根, 而且是一个非本原 4 次单位根. 容易验证, 对任意整数 n, 复数 $\omega_n = e^{-2\pi i/n}$ 是一个本原 n 次单位根. 数 $e^{2\pi i/n}$ 也是一个本原 n 次单位根, 但是我们将按通常的习惯用前者作为 Fourier 变换的基. 图 10-3 表示一个本原 8 次单位根 $\omega_8 = e^{-2\pi i/8}$ 和其余 7 个单位根, 它们都是 ω_8 的幂.

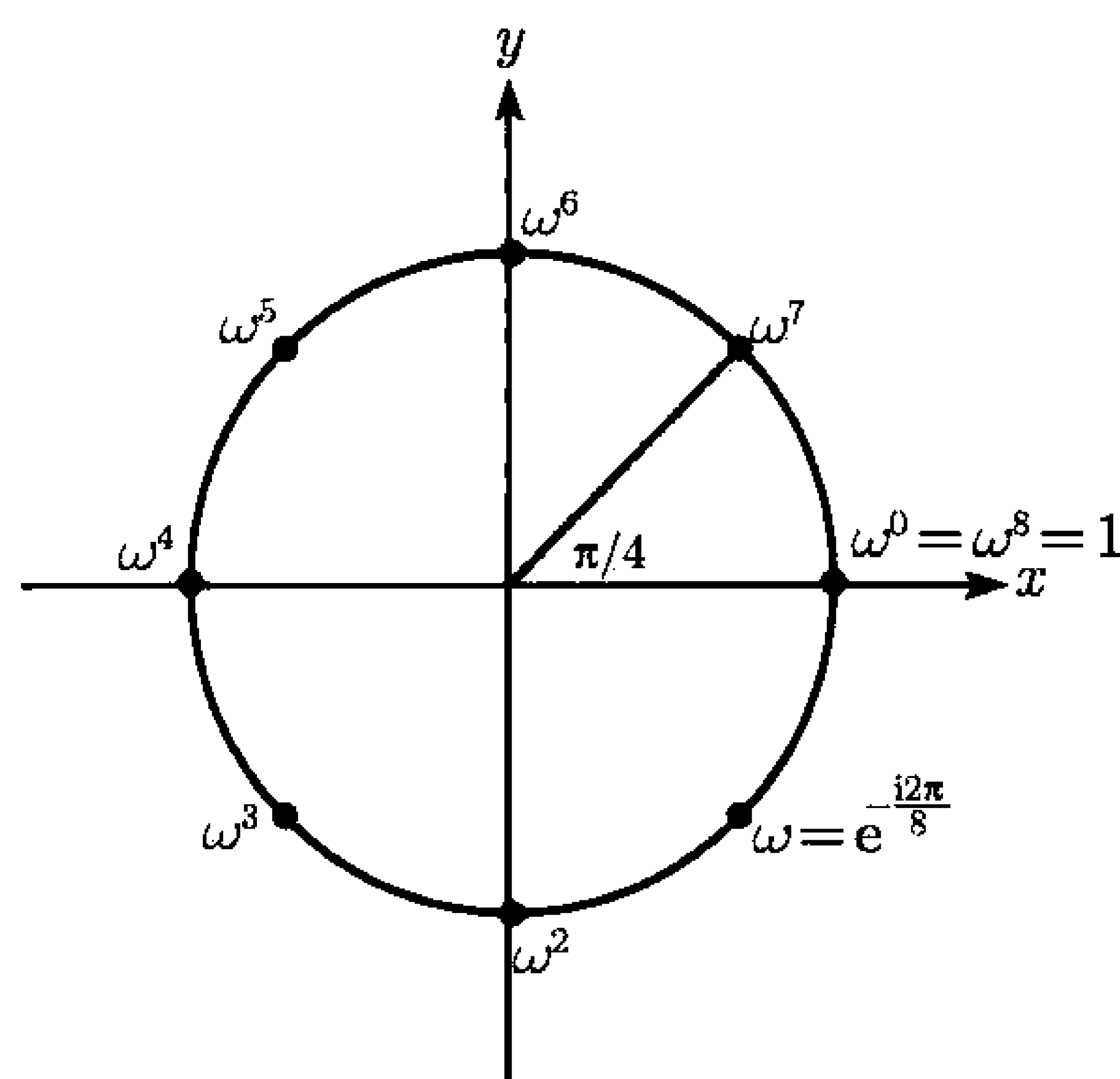


图 10-3 图中表示 8 个 8 次单位根. 它们由 $\omega = e^{-2\pi i/8}$ 生成, 意即对某个 k, 每一个都是 ω^k. 尽管 ω 和 ω^3 是本原 8 次单位根, 但是 ω^2 不是, 这是因为它也是 4 次单位根

这里有一个关键的恒等式, 我们以后在简化离散 Fourier 变换的计算中将需要它, 当 $n > 1$ 时, 用 ω 表示 n 次单位根 $\omega = e^{-2\pi i/n}$, 则有

$$1 + \omega + \omega^2 + \omega^3 + \cdots + \omega^{n-1} = 0. \tag{10.3}$$

这个等式可以通过下面的伸缩和 (telescoping sum) 证得

$$(1-\omega)(1 + \omega + \omega^2 + \omega^3 + \cdots + \omega^{n-1}) = 1 - \omega^n = 0. \tag{10.4}$$

因为左边第一项不是零, 所以第二项必须是零. 使用同样的方法可以证明

$$\begin{aligned}
&1 + \omega^2 + \omega^4 + \omega^6 + \cdots + \omega^{2(n-1)} = 0, \\
&1 + \omega^3 + \omega^6 + \omega^9 + \cdots + \omega^{3(n-1)} = 0, \\
&\qquad\cdots\cdots\cdots\cdots \\
&1 + \omega^{n-1} + \omega^{2(n-1)} + \omega^{3(n-1)} + \cdots + \omega^{(n-1)(n-1)} = 0.
\end{aligned} \tag{10.5}$$

最后一个等式与以上不同, 是

$$1+\omega^n+\omega^{2n}+\omega^{3n}+\cdots+\omega^{n(n-1)}=1+1+1+1+\cdots+1=n. \tag{10.6}$$

依据以上这些信息, 可得以下引理.

引理 10.1 (本原单位根) 设 ω 是一个本原 n 次单位根, k 是整数, 那么

$$\sum_{j=0}^{n-1}\omega^{jk}=\begin{cases} n, & \text{如果 } \frac{k}{n} \text{ 是整数}, \\ 0, & \text{否则}. \end{cases}$$

习题 6 要求读者给出证明的细节.

10.1.2 离散 Fourier 变换

设 $\boldsymbol{x}=[x_0,\cdots,x_{n-1}]^{\mathrm{T}}$ 是一个 (实值) n 维向量, 记为 $\omega=\mathrm{e}^{-2\pi\mathrm{i}/n}$. 下面是本章的基本定义.

定义 10.2 $\boldsymbol{x}=[x_0,\cdots,x_{n-1}]^{\mathrm{T}}$ 的离散 Fourier 变换 (DFT) 是 n 维向量 $\boldsymbol{y}=[y_0,\cdots,y_{n-1}]$, 这里 $\omega=\mathrm{e}^{-2\pi\mathrm{i}/n}$,

$$y_k=\frac{1}{\sqrt{n}}\sum_{j=0}^{n-1}x_j\omega^{jk}. \tag{10.7}$$

例如, 引理 10.1 表明, $\boldsymbol{x}=[1,1,\cdots,1]$ 的 DFT 是 $\boldsymbol{y}=[\sqrt{n},0,\cdots,0]$. 用矩阵表示, 这个定义表明

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix}=\begin{bmatrix} a_0+\mathrm{i}b_0 \\ a_1+\mathrm{i}b_1 \\ a_2+\mathrm{i}b_2 \\ \vdots \\ a_{n-1}+\mathrm{i}b_{n-1} \end{bmatrix}=\frac{1}{\sqrt{n}}\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ \omega^0 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \omega^0 & \omega^3 & \omega^6 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}. \tag{10.8}$$

每一个 $y_k=a_k+\mathrm{i}b_k$ 是一个复数.(10.8) 中的 $n\times n$ 矩阵叫做**Fourier 矩阵**

$$\boldsymbol{F}_n=\frac{1}{\sqrt{n}}\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ \omega^0 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \omega^0 & \omega^3 & \omega^6 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{bmatrix}. \tag{10.9}$$

除了第一行外, Fourier 矩阵的每一行加起来都等于零. 因为 $\boldsymbol{F}_n$ 是对称矩阵, 所以对于列也同样如此. Fourier 矩阵有显式逆矩阵

$$\boldsymbol{F}_n^{-1}=\frac{1}{\sqrt{n}}\begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ \omega^0 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \omega^0 & \omega^{-3} & \omega^{-6} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega^0 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)^2} \end{bmatrix}, \tag{10.10}$$

并且向量 $\boldsymbol{y}$ 的**逆离散 Fourier 变换**是 $\boldsymbol{x}=\boldsymbol{F}_n^{-1}\boldsymbol{y}$. (10.10) 是矩阵 $\boldsymbol{F}_n$ 的逆矩阵, 验证这一点需要有关 n 次单位根的引理 10.1. 见习题 8.

设 $z=\mathrm{e}^{\mathrm{i}\theta}=\cos\theta+\mathrm{i}\sin\theta$ 是单位圆上的一点. 那么它的倒数 $\mathrm{e}^{-\mathrm{i}\theta}=\cos\theta-\mathrm{i}\sin\theta$ 是它的复共轭. 因此, 逆 DFT 矩阵的元素是 $\boldsymbol{F}_n$ 的元素的复共轭:

$$\boldsymbol{F}_n^{-1}=\bar{\boldsymbol{F}}_n. \tag{10.11}$$

定义 10.3 *复向量 $\boldsymbol{v}$ 的***度量** (magnitude) *是实数 $\|\boldsymbol{v}\|=\sqrt{\bar{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{v}}$. 如果 $\bar{\boldsymbol{F}}^{\mathrm{T}}\boldsymbol{F}=\boldsymbol{I}$, 那么复矩阵 $\boldsymbol{F}$ 是***酉矩阵** (unitary).

像 Fourier 矩阵一样, 酉矩阵是实正交矩阵的复形式. 如果 $\boldsymbol{F}$ 是酉矩阵, 那么 $\|\boldsymbol{F}\boldsymbol{v}\|^2=\bar{\boldsymbol{v}}^{\mathrm{T}}\bar{\boldsymbol{F}}^{\mathrm{T}}\boldsymbol{F}\boldsymbol{v}=\bar{\boldsymbol{v}}^{\mathrm{T}}\boldsymbol{v}=\|\boldsymbol{v}\|^2$. 因此, 在这种情况下, 一个向量左乘 $\boldsymbol{F}$ 或者 $\boldsymbol{F}^{-1}$, 它的度量不变.

使用离散 Fourier 变换只是用 $n\times n$ 矩阵 $\boldsymbol{F}_n$ 相乘的问题, 因此需要 $O(n^2)$ 次运算 [明确地讲, 是 n^2 次乘法和 $n(n-1)$ 次加法]. 用 $\boldsymbol{F}_n^{-1}$ 相乘的逆离散 Fourier 变换也是 $O(n^2)$ 次运算. 在 10.1.3 节, 我们建立了可以明显减少运算次数的 DFT 形式, 称为快速 Fourier 变换.

例 10.1 求向量 $\boldsymbol{x}=[1,0,-1,0]^{\mathrm{T}}$ 的 DFT.

设 ω 是 4 次单位根, 或者 $\omega=\mathrm{e}^{-\mathrm{i}\pi/2}=\cos(\frac{\pi}{2})-\mathrm{i}\sin(\frac{\pi}{2})=-\mathrm{i}$. 应用 DFT, 我们得到

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}=\frac{1}{\sqrt{4}}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix}=\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix}=\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}. \tag{10.12}$$

◀

MATLAB 命令 `fft` 以稍为不同的标准来执行 DFT, 所以 $\boldsymbol{F}_n\boldsymbol{x}$ 是由 `fft`(x)/`sqrt`(n)来计算的. 逆命令 `ifft` 是 `fft` 的逆. 因此, $\boldsymbol{F}_n^{-1}\boldsymbol{y}$ 是由 MATLAB命令

ifft(y)*sqrt(n) 来计算的. 换言之, MATLAB 的 fft 和 iff 是互逆的. 尽管它们的标准与这里给出的定义不同, 它们具有 $\boldsymbol{F}_n$ 和 $\boldsymbol{F}_n^{-1}$ 是酉矩阵的优点.

即使向量有实数分量, 也没有理由使 $\boldsymbol{y}$ 的分量是实数. 但是, 如果 x_j 是实数, 那么复数 y_k 有以下特殊的性质.

引理 10.4　设 $\{y_k\}$ 是 $\{x_j\}$ 的 DFT, 这里 x_j 是实数, 那么 (a) y_0 是实数, (b) $y_{n-k} = \bar{y}_k(k = 1, \cdots, n-1)$.

证　根据 DFT 的定义 (10.7), (a) 的理由很清楚. 这是因为 y_0 是 x_j 的和除以 $\sqrt{n}$. 结论 (b) 是根据以下事实得出的:

$$\omega^{n-k} = \mathrm{e}^{-2\pi\mathrm{i}(n-k)/n} = \mathrm{e}^{-2\pi\mathrm{i}}\mathrm{e}^{2\pi\mathrm{i}k/n} = \cos(2\pi k/n) + \mathrm{i}\sin(2\pi k/n)$$

其中

$$\omega^k = \mathrm{e}^{-2\pi\mathrm{i}k/n} = \cos(2\pi k/n) - \mathrm{i}\sin(2\pi k/n).$$

这就意味着 $\omega^{n-k} = \overline{\omega^k}$, 从 Fourier 变换的定义,

$$y_{n-k} = \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1} x_j(\omega^{n-k})^j = \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1}\overline{x_j(\omega^k)^j} = \frac{1}{\sqrt{n}}\sum_{j=0}^{n-1}\overline{x_j(\omega^k)^j} = \bar{y}_k.$$

这里我们用了复共轭的乘积就是乘积的共轭这个事实.

引理 10.4 有一个有趣的结论. 设 n 是偶数, 并且 $x_0, \cdots, x_{n-1}$ 是实数, 那么 Fourier 变换 DFT 把它们正好代替成 n 个其他的实数 $a_0, a_1, b_1, a_2, b_2, \cdots, a_{\frac{n}{2}}$, 即 Fourier 变换 $y_0, \cdots, y_{n-1}$ 的实部和虚部. 举例来说, $n = 8$ 的 DFT 有如下形式:

$$\boldsymbol{F}_8\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 + \mathrm{i}b_1 \\ a_2 + \mathrm{i}b_2 \\ a_3 + \mathrm{i}b_3 \\ a_4 \\ a_3 - \mathrm{i}b_3 \\ a_2 - \mathrm{i}b_2 \\ a_1 - \mathrm{i}b_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_{\frac{n}{2}-1} \\ y_{\frac{n}{2}} \\ \bar{y}_{\frac{n}{2}-1} \\ \vdots \\ \bar{y}_1 \end{bmatrix}. \tag{10.13}$$

10.1.3　快速 Fourier 变换

如 10.1.2 节提到的, 以传统的方式对 n 维向量用离散 Fourier 变换需要 $O(n^2)$ 次运算, Cooley 和 Tukey[5] 找到一种只需要 $O(n\log n)$ 次运算来完成 DFT 的方法, 这种算法称为**快速 Fourier 变换** (FFT). 对数据分析应用 FFT 几乎立刻流行开来. 信号处理领域由于使用该算法将原先的模拟信号转化为数字信号. 我们将解释这种方法并且通过运算次数来说明它相对于原本的 DFT(10.8) 的优越性.

亮点 复杂性

Cooley 和 Tukey 把 DFT 的复杂性从 $O(n^2)$ 次运算减少到 $O(n \ln n)$ 次运算, 这一成就促进了 Fourier 变换方法的广泛应用. 与问题本身的大小 “几乎线性地” 成比例的方法是很重要的. 例如, 对于实时数据就有使用它的可能性. 这是因为分析大致能够出现在需要数据的同一时间尺度处. 不久以后人们用特殊的电路来实现 FFT, 现在则是用 DSP 芯片来表示 FFT 这种芯片广泛应用于分析和控制电子系统中.

可以把 DFT $\boldsymbol{F}_n\boldsymbol{x}$ 写成

$$\begin{bmatrix} y_0 \\ \vdots \\ y_{n-1} \end{bmatrix} = \frac{1}{\sqrt{n}}\boldsymbol{M}_n \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix},$$

这里

$$\boldsymbol{M}_n = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ \omega^0 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \omega^0 & \omega^3 & \omega^6 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{bmatrix}.$$

我们将说明如何递推地计算 $\boldsymbol{z} = \boldsymbol{M}_n\boldsymbol{x}$. 要完成 DFT, 只需要除以 $\sqrt{n}$, 或者 $\boldsymbol{y} = \boldsymbol{F}_n\boldsymbol{x} = \boldsymbol{z}/\sqrt{n}$.

我们首先说明 $n = 4$ 这种情形, 来得到大概的主要思想. 然后, 推广到一般情形. 设 $\omega = \mathrm{e}^{-2\pi \mathrm{i}/4} = -\mathrm{i}$. 离散 Fourier 变换是

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{10.14}$$

写出矩阵乘积, 但要重新排列这些项的阶, 使得偶数阶的项在前面:

$$\begin{aligned} z_0 &= \omega^0 x_0 + \omega^0 x_2 + \omega^0(\omega^0 x_1 + \omega^0 x_3), \\ z_1 &= \omega^0 x_0 + \omega^2 x_2 + \omega^1(\omega^0 x_1 + \omega^2 x_3), \\ z_2 &= \omega^0 x_0 + \omega^4 x_2 + \omega^2(\omega^0 x_1 + \omega^4 x_3), \\ z_3 &= \omega^0 x_0 + \omega^6 x_2 + \omega^3(\omega^0 x_1 + \omega^6 x_3). \end{aligned}$$

利用 $\omega^4=1$ 这个事实, 可以把这些等式写成

$$\begin{aligned}
z_0&=(\omega^0x_0+\omega^0x_2)+\omega^0(\omega^0x_1+\omega^0x_3),\\
z_1&=(\omega^0x_0+\omega^2x_2)+\omega^1(\omega^0x_1+\omega^2x_3),\\
z_2&=(\omega^0x_0+\omega^0x_2)+\omega^2(\omega^0x_1+\omega^0x_3),\\
z_3&=(\omega^0x_0+\omega^2x_2)+\omega^3(\omega^0x_1+\omega^2x_3).
\end{aligned}$$

注意在前面两行中圆括号中的每一项都与底下两行的相同. 定义

$$u_0=\mu^0x_0+\mu^0x_2,\qquad u_1=\mu^0x_0+\mu^1x_2,$$

$$v_0=\mu^0x_1+\mu^0x_3,\qquad v_1=\mu^0x_1+\mu^1x_3,$$

这里 $\mu=\omega^2$ 是一个二次单位根, $\boldsymbol{u}=(u_0,u_1)^{\mathrm{T}}$ 及 $\boldsymbol{v}=(v_0,v_1)^{\mathrm{T}}$ 实质上都是 $n=2$ 的 DFT; 更精确地讲,

$$\boldsymbol{u}=\boldsymbol{M}_2\begin{bmatrix}x_0\\x_2\end{bmatrix},\qquad \boldsymbol{v}=\boldsymbol{M}_2\begin{bmatrix}x_1\\x_3\end{bmatrix}.$$

可以把原来的 $\boldsymbol{M}_4\boldsymbol{x}$ 写成

$$\begin{aligned}
z_0&=u_0+\omega^0v_0,\\
z_1&=u_1+\omega^1v_1,\\
z_2&=u_0+\omega^2v_0,\\
z_3&=u_1+\omega^3v_1.
\end{aligned}$$

总之, DFT (4) 的计算已被缩减为一组 DFT(2) 的计算再加上某些额外乘法和加法.

暂时不考虑 $\frac{1}{\sqrt{n}}$, DFT(n) 能够被缩减为计算两个 DFT($\frac{n}{2}$) 再加上 $2n-1$ 次额外运算 ($n-1$ 次乘法及 n 次加法). 仔细计算这些必要的加法和乘法, 得到定理 10.5.

定理 10.5 (FFT 的运算次数)　*设 n 是 2 的幂. 规格为 n 的快速 Fourier 变换可以用 $n(2\log_2 n-1)+1$ 次加法和乘法以及一次除法 (除以 $\sqrt{n}$) 来完成.*

证　不考虑在最后要用到的平方根. 定理的结果等价于: DFT (2^m) 能够用 $2^m(2m-1)+1$ 次加法和乘法来完成. 事实上, 当 n 是偶数时, 我们从上面看到如何把 DFT(n) 化成一对 DFT ($\frac{n}{2}$). 如果 n 是 2 的幂 (譬如说 $n=2^m$), 那么可以递推地分解这个问题直到我们得到 DFT (1), 这是用 1×1 单位矩阵相乘, 用了零次运算. 从最低开始, DFT (1) 无须进行运算, 而 DFT (2) 需要两次加法和一次乘法: $y_0=u_0+1v_0, y_1=u_0+\omega v_0$, 这里 u_0 和 v_0 是 DFT (1) (即 $u_0=y_0, v_0=y_1$).

DFT (4) 需要两次 DFT (2) 加上 $2\times4-1=7$ 次进一步的运算, 总共需要 $2(3)+7=2^m(2m-1)+1$ 次运算, 这里 $m=2$. 我们通过归纳继续: 假设对给

定的 m, 这个公式是正确的, 那么 DFT(2^{m+1}) 需要用两个 DFT (2^m), 它要进行 $2[2^m(2m-1)+1]$ 次运算, 加上 $2\times 2^{m+1}-1$ 次额外的运算 [完成类似于 (10.15) 这样的等式], 总共是

$$2[2^m(2m-1)+1]+2^{m+2}-1=2^{m+1}(2m-1+2)+2-1=2^{m+1}[(2m+1)-1]+1.$$

因此, 对于 DFT (2^m) 的快速形式证得运算次数的公式 $2^m(2m-1)+1$. 由此, 得到结论.

不必做更多的工作, 就能够用 DFT 的快速算法来构造逆 DFT 的快速算法. 逆 DFT 是复共轭矩阵 $\bar{\boldsymbol{F}}_n$. 要执行复向量 $\boldsymbol{y}$ 的逆 DFT, 只用取共轭, 利用 FFT, 并对结果取共轭, 这是因为

$$\boldsymbol{F}_n^{-1}\boldsymbol{y}=\bar{\boldsymbol{F}}_n\boldsymbol{y}=\overline{\boldsymbol{F}_n\bar{\boldsymbol{y}}}. \tag{10.15}$$

习题 10.1

1. 求以下向量的 DFT:
   (a) $[0,1,0,-1]$; (b) $[1, 1, 1, 1]$; (c) $[0,-1,0,1]$; (d) $[0,1,0,-1,0,1,0,-1]$.
2. 求以下向量的 DFT:
   (a) $\left[\frac{3}{4},\frac{1}{4},\frac{-1}{4},\frac{1}{4}\right]$; (b) $\left[\frac{9}{4},\frac{1}{4},-\frac{3}{4},\frac{1}{4}\right]$; (c) $\left[1,0,-\frac{1}{2},0\right]$;
   (d) $\left[1,0,-\frac{1}{2},0,1,0,-\frac{1}{2},0\right]$.
3. 求以下向量的逆 DFT:
   (a) $[1,0,0,0]$; (b) $[1,1,-1,1]$; (c) $[1,-\mathrm{i},1,\mathrm{i}]$; (d) $[1,0,0,0,3,0,0,0]$.
4. 求以下向量的逆 DFT:
   (a) $[0,-\mathrm{i},0,\mathrm{i}]$; (b) $[2,0,0,0]$; (c) $\left[\frac{1}{2},\frac{1}{2},0,\frac{1}{2}\right]$; (d) $\left[1,\frac{3}{2},\frac{1}{2},\frac{3}{2}\right]$.
5. (a) 写出所有的 4 次单位根及所有的本原 4 次单位根. (b) 写出所有本原 7 次单位根. (c) 对于素数 p, 存在多少个本原 p 次单位根?
6. 证明引理 10.1.
7. 对习题 1 中的 Fourier 变换, 求如在 (10.13) 中的实数 $a_0,a_1,b_1,a_2,b_2,\cdots,a_{\frac{n}{2}}$.
8. 证明 (10.10) 中的矩阵是 Fourier 矩阵 $\boldsymbol{F}_n$ 的逆.

10.2 三角插值

离散 Fourier 变换实际上能做什么? 本节提出解释把 Fourier 变换的输出向量 $\boldsymbol{y}$ 当作等距间隔的数据的插值系数, 以便更好地理解它的工作.

10.2.1 DFT 插值定理

设 $[c,d]$ 是一个区间, n 是一个正整数. 定义 $\Delta t=(d-c)/n$, 以及 $t_j=c+$

$j\Delta t\,(j=0,\cdots,n-1)$ 是区间中等距间隔的点. 对于 Fourier 变换给定的输入向量 $\boldsymbol{x}$, 我们将把分量 x_j 解释为所测量信号的第 j 个分量. 例如, 可以把 $\boldsymbol{x}$ 的分量想象成一系列在离散的等距间隔时间 t_j 处测得的度量, 如图 10-4 所示.

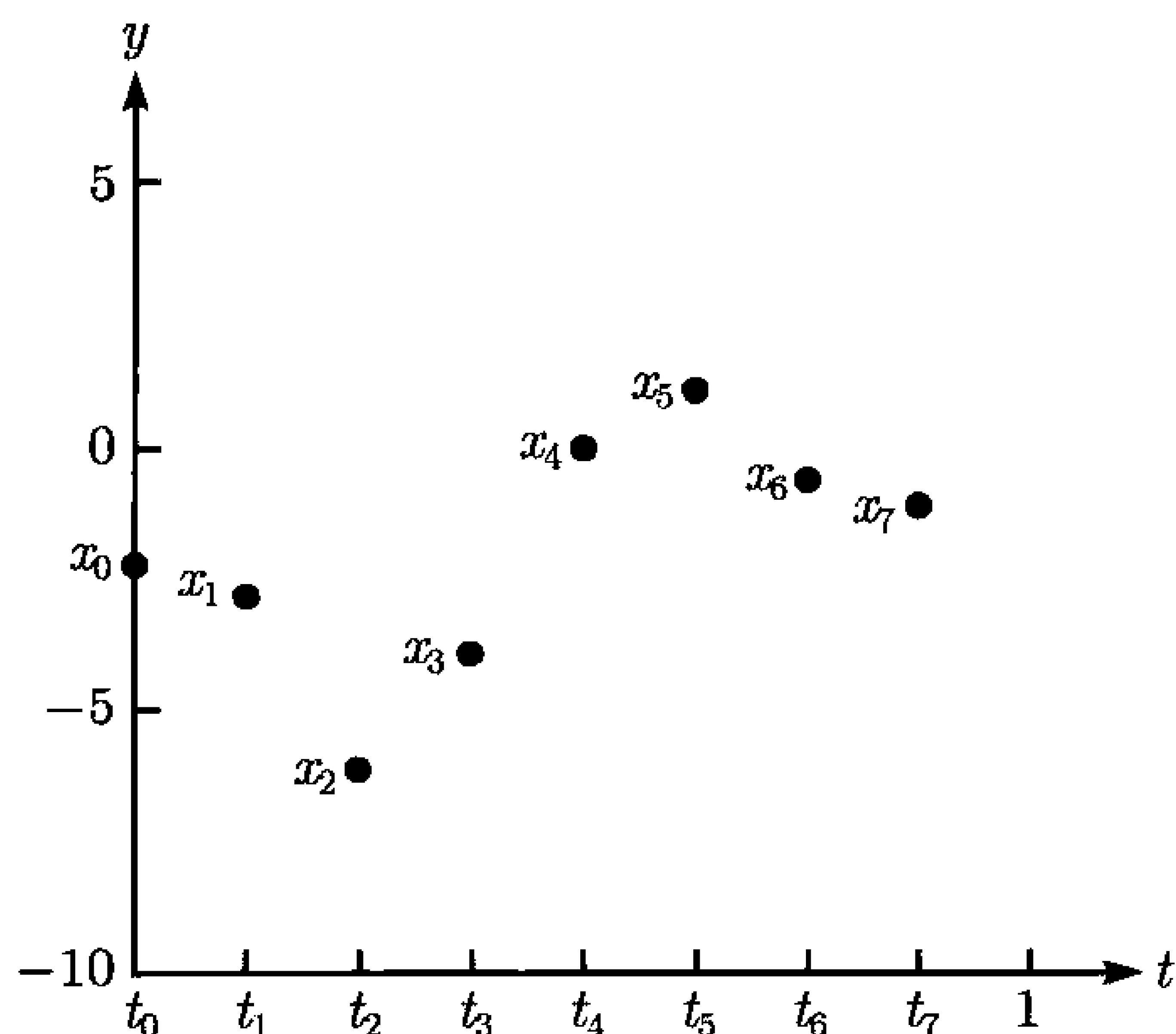


图 10-4　x 的分量看作时间序列. Fourier 变换是一种计算插值了这种数据的三角多项式的方法

设 $\boldsymbol{y}=\boldsymbol{F}_n\boldsymbol{x}$ 是 $\boldsymbol{x}$ 的 DFT. 因为 $\boldsymbol{x}$ 是 $\boldsymbol{y}$ 的逆 DFT, 所以我们可以从 (10.10) 写出 $\boldsymbol{x}$ 的分量的显式公式, 记住 $\omega=\mathrm{e}^{-2\pi\mathrm{i}/n}$:

$$x_j=\frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}y_k(\omega^{-k})^j=\frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}y_k\mathrm{e}^{2\pi kj\mathrm{i}/n}=\sum_{k=0}^{n-1}y_k\frac{\mathrm{e}^{\frac{2\pi k(t_j-c)\mathrm{i}}{d-c}}}{\sqrt{n}}. \tag{10.16}$$

我们能够把它看成点 (t_j,x_j) 用三角基函数的插值, 这里的系数是 y_k. 定理 10.6 是 (10.16) 的简单复述, 表明用基函数 $\mathrm{e}^{\mathrm{i}2\pi k(t-c)/(d-c)}/\sqrt{n}\,(k=0,\cdots,n-1)$ 对数据点 (t_j,x_j) 进行插值, 其插值系数由 $\boldsymbol{F}_n\boldsymbol{x}$ 给出.

定理 10.6 (DFT 插值定理)　给定区间 $[c,d]$ 及正整数 n. 设 $t_j=c+j(d-c)/n(j=0,\cdots,n-1)$, 并设 $\boldsymbol{x}=(x_0,\cdots,x_{n-1})$ 表示 n 个数的向量. 定义 $\vec{a}+\vec{b}\mathrm{i}=\boldsymbol{F}_n\boldsymbol{x}$, 这里 $\boldsymbol{F}_n$ 是离散 Fourier 变换矩阵, 那么复函数

$$Q(t)=\frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}(a_k+\mathrm{i}b_k)\mathrm{e}^{2\pi k(t-c)\mathrm{i}/(d-c)}$$

满足 $Q(t_j)=x_j(j=0,\cdots,n-1)$. 而且, 如果 x_j 是实数, 那么实函数

$$P(t)=\frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}\left(a_k\cos\frac{2\pi k(t-c)}{d-c}-b_k\sin\frac{2\pi k(t-c)}{d-c}\right)$$

满足 $P(t_j)=x_j,j=0,\cdots,n-1$.

换言之, Fourier 变换 F_n 把数据 $\{x_j\}$ 变换成插值系数.

该定理的最后部分的解释是, 利用 Euler 公式, 我们可以把插值函数 (10.16) 重新写成

$$Q(t)=\frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}(a_k+\mathrm{i}b_k)\left(\cos\frac{2\pi k(t-c)}{d-c}+\mathrm{i}\sin\frac{2\pi k(t-c)}{d-c}\right).$$

把插值函数 $Q(t)=P(t)+\mathrm{i}I(t)$ 分成实部和虚部. 因为 x_j 是实数, 只有 $Q(t)$ 的实部需要插值 x_j. 实部是

$$P(t)=P_n(t)=\frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}\left(a_k\cos\frac{2\pi k(t-c)}{d-c}-b_k\sin\frac{2\pi k(t-c)}{d-c}\right). \tag{10.17}$$

这里的下标 n 表示这个三角型式中的项数. 我们有时将称 P_n 是n **阶三角函数**. 引理 10.4 及下面的引理 10.7 可以用来进一步简化插值函数 $P_n(t)$.

引理 10.7 设 $t=\frac{j}{n}$, 这里 j 和 n 都是整数. 设 k 是整数, 那么

$$\cos 2(n-k)\pi t=\cos 2k\pi t,\quad \sin 2(n-k)\pi t=-\sin 2k\pi t. \tag{10.18}$$

事实上, 由余弦加法公式得到 $\cos 2(n-k)\pi j/n=\cos(2\pi j-2jk\pi/n)=\cos(-2jk\pi/n)$. 对正弦有类似的结论.

引理 10.7 和引理 10.4 一起表明, 三角展开式 (10.17) 的后半部分是多余的. 我们可以只用前半部分的项在 t_j 处插值 (除了改变正弦项的符号). 根据引理 10.4, 展开式的后半部分的系数与前半部分的系数相同 (除了改变正弦项的符号). 于是符号的改变使得正弦项互相抵消, 这样我们已经证明了 P_n 的简化形式是

$$P_n(t)=\frac{a_0}{\sqrt{n}}+\frac{2}{\sqrt{n}}\sum_{k=1}^{n/2-1}\left(a_k\cos\frac{2k\pi(t-c)}{d-c}-b_k\sin\frac{2k\pi(t-c)}{d-c}\right)+\frac{a_{n/2}}{\sqrt{n}}\cos\frac{n\pi(t-c)}{d-c}.$$

为了写这一个表达式, 我们已经假定 n 是偶数. 对 n 是奇数, 公式稍有不同. 见习题 7.

推论 10.8 对于偶数 n, 令 $t_j=c+j(d-c)/n$ $(j=0,\cdots,n-1)$, 并设 $\boldsymbol{x}=(x_0,\cdots,x_{n-1})$ 表示 n 个实数的向量. 定义 $\vec{a}+\vec{b}\mathrm{i}=\boldsymbol{F}_n\boldsymbol{x}$, 这里 $\boldsymbol{F}_n$ 是离散 Fourier 变换, 那么函数

$$P_n(t)=\frac{a_0}{\sqrt{n}}+\frac{2}{\sqrt{n}}\sum_{k=1}^{n/2-1}\left(a_k\cos\frac{2k\pi(t-c)}{d-c}-b_k\sin\frac{2k\pi(t-c)}{d-c}\right)+\frac{a_{n/2}}{\sqrt{n}}\cos\frac{n\pi(t-c)}{d-c} \tag{10.19}$$

满足 $P_n(t_j)=x_j, j=0,\cdots,n-1$.

例 10.2　对例 10.1, 求三角插值.

区间是 $[c,d]=[0,1]$. 设 $\boldsymbol{x}=[1,0,-1,0]^{\mathrm{T}}$, 并且计算它的 DFT 是 $\boldsymbol{y}=[0,1,0,1]^{\mathrm{T}}$. 插值系数是 $a_k+\mathrm{i}b_k=y_k$. 因此, $a_0=a_2=0, a_1=a_3=1, b_0=b_1=b_2=b_3=0$. 根据 (10.19), 我们只需要 a_0,a_1,a_2 和 b_1.$\boldsymbol{x}$ 的三角插值函数由下式给出:

$$P_4(t)=\frac{a_0}{2}+(a_1\cos 2\pi t-b_1\sin 2\pi t)+\frac{a_2}{2}\cos 4\pi t=\cos 2\pi t.$$

点 (t,x) 的插值如图 10-5 所示, 这里 $t=[0,\frac{1}{4},\frac{1}{2},\frac{3}{4}]$, $\boldsymbol{x}=[1,0,-1,0]$. ◀

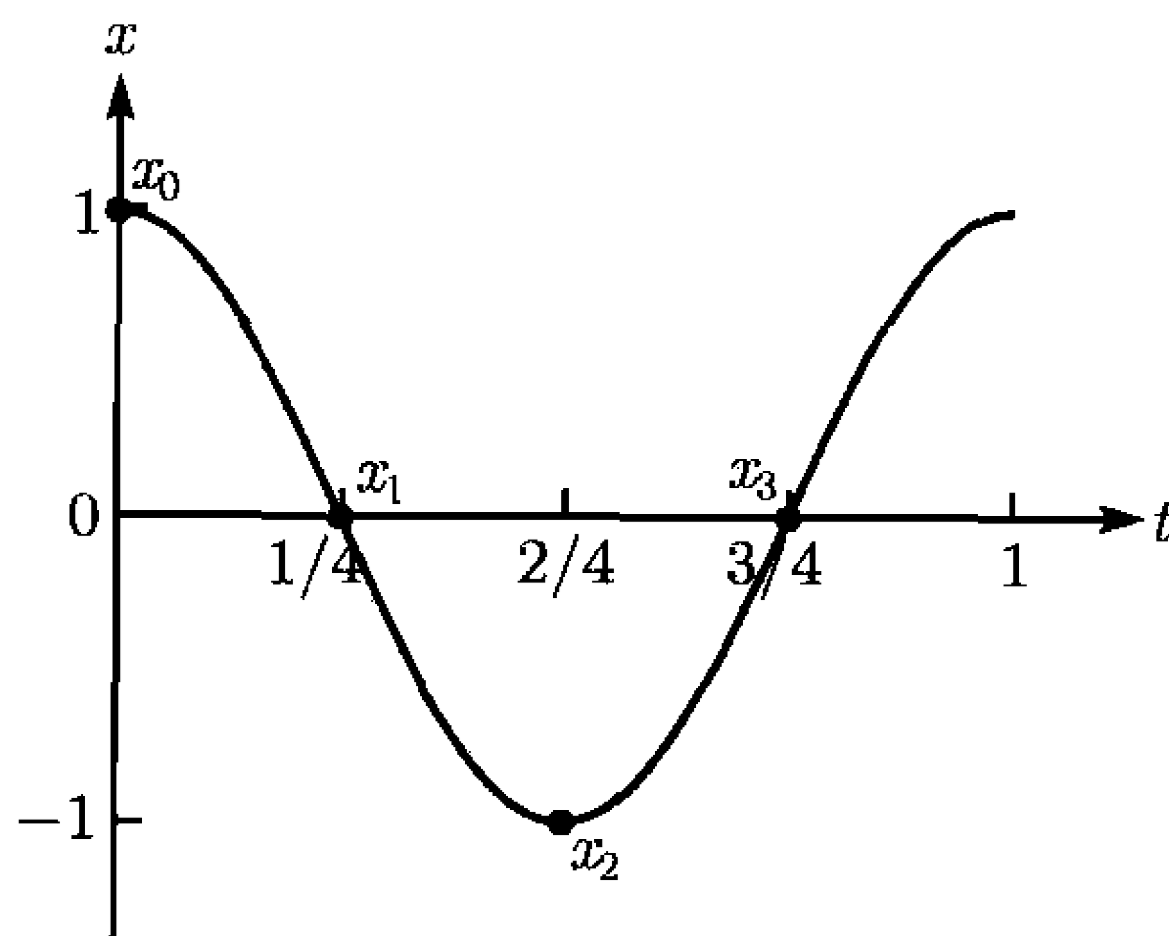


图 10-5　三角插值. 输入向量 $\boldsymbol{x}$ 是 $[1,0,-1,0]^{\mathrm{T}}$, 公式 (10.19) 给出的插值函数是 $P_4(t)=\cos 2\pi t$

例 10.3　对于例 4.6 中的温度数据 $\boldsymbol{x}=[-2.2,-2.8,-6.1,-3.9,0.0,1.1,-0.6,-1.1]$, 求在区间 $[0,1]$ 上的三角插值.

Fourier 变换输出, 精确到 4 位小数是

$$\boldsymbol{y}=\begin{bmatrix}-5.515\,4\\-1.052\,8+3.619\,5\mathrm{i}\\1.591\,0-1.166\,7\mathrm{i}\\-0.502\,8-0.269\,5\mathrm{i}\\-0.777\,8\\-0.502\,8+0.269\,5\mathrm{i}\\1.591\,0+1.166\,7\mathrm{i}\\-1.052\,8+3.619\,5\mathrm{i}\end{bmatrix}.$$

按照公式 (10.19), 插值函数是

$$P_8(t)=\frac{-5.515\,4}{\sqrt{8}}-\frac{1.052\,8}{\sqrt{2}}\cos 2\pi t-\frac{3.619\,5}{\sqrt{2}}\sin 2\pi t+\frac{1.591\,0}{\sqrt{2}}\cos 4\pi t+\frac{1.166\,7}{\sqrt{2}}\sin 4\pi t$$

$$-\frac{0.502\ 8}{\sqrt{2}}\cos 6\pi t+\frac{0.269\ 5}{\sqrt{2}}\sin 6\pi t-\frac{0.777\ 8}{\sqrt{8}}\cos 8\pi t$$

$$=-1.95-0.744\ 5\cos 2\pi t-2.559\ 4\sin 2\pi t+1.125\cos 4\pi t+0.825\sin 4\pi t$$

$$-0.355\ 5\cos 6\pi t+0.190\ 6\sin 6\pi t-0.275\ 0\cos 8\pi t. \tag{10.20}$$

图 10-6 显示了数据点以及三角插值函数. ◀

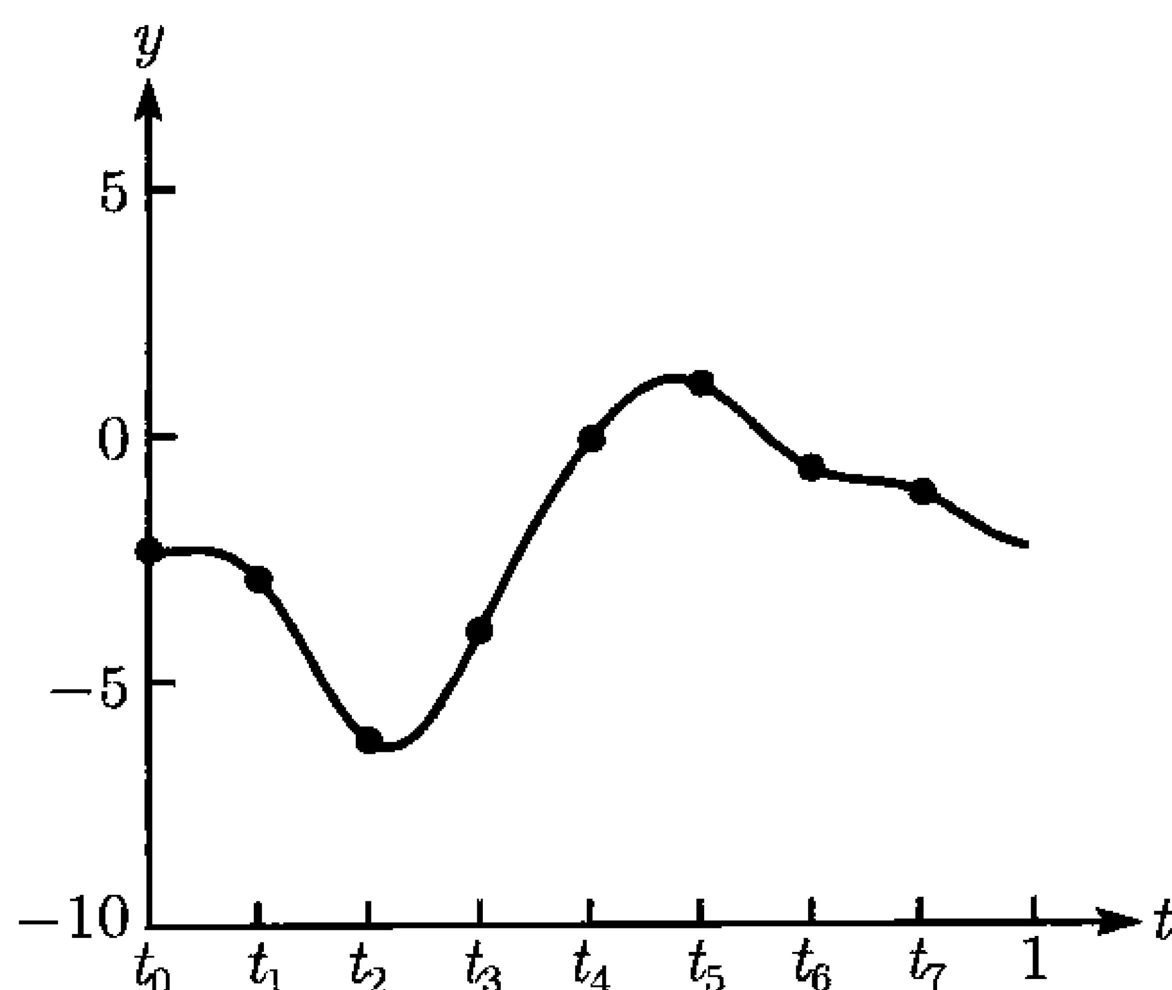


图 10-6 例 4.6 中数据的三角插值. 使用 $n=8$ 的 Fourier 变换来插值数据 $\boldsymbol{t}=[0,\frac{1}{8},\frac{1}{4},\frac{3}{8},\frac{1}{2},\frac{5}{8},\frac{3}{4},\frac{7}{8}]$, $\boldsymbol{x}=[-2.2,-2.8,-6.1,-3.9,0.0,1.1,-0.6,-1.1]$. 用程序 10.1 取 $p=100$ 制作这个图

10.2.2 三角函数的有效求值

推论 10.8 是关于插值的重要声明. 尽管初看起来它似乎复杂, 但有另一种方法来求值和绘制图 10-5 和图 10-6 中的三角插值多项式, 即用 DFT 来做所有的工作而不是绘制 (10.19) 中的正弦和余弦曲线. 毕竟我们从定理 10.6 知道, 把数据向量 $\boldsymbol{x}$ 乘以 $\boldsymbol{F}_n$ 就把数据变成插值系数. 反过来, 我们可以把插值系数转化成数据点. 不是求 (10.19) 的值, 仅仅是把 DFT 倒过来: 把插值系数 $\{a_k+\mathrm{i}b_k\}$ 的向量乘以 $\boldsymbol{F}_n^{-1}$.

当然, 如果在运算 $\boldsymbol{F}_n$ 之后运算它的逆 $\boldsymbol{F}_n^{-1}$, 我们就只是回到了原来的数据点. 恰恰相反, 我们将设 $p\geqslant n$ 是一个更大的数. 我们打算把 (10.19) 看作一个 p 阶三角函数, 然后把 Fourier 变换反过来求曲线在 p 个等距点处的值. 可以把 p 取得足够大以得到看似连续的图.

把 $P_n(t)$ 的系数看成一个 p 阶三角多项式的系数, 注意, 我们可以把 (10.19) 重新写成

$$P_p(t)=\frac{\sqrt{\frac{p}{n}}a_0}{\sqrt{p}}+\frac{2}{\sqrt{p}}\sum_{k=1}^{p/2-1}\left(\sqrt{\frac{p}{n}}a_k\cos\frac{2k\pi(t-c)}{d-c}-\sqrt{\frac{p}{n}}b_k\sin\frac{2k\pi(t-c)}{d-c}\right)$$

$$+\frac{\sqrt{\frac{p}{n}}a_{n/2}}{\sqrt{p}}\cos n\pi t, \tag{10.21}$$

这里置 $a_k = b_k = 0$ $(k = \frac{n}{2}+1,\cdots,\frac{p}{2})$. 我们从 (10.21) 得出结论：产生落在 $t_j = c + j(d-c)/n$ $(j = 0,\cdots,n-1)$ 的曲线 (10.19) 的 p 个点的方法就是, 用 $\sqrt{\frac{p}{n}}$ 乘 Fourier 系数然后再求 DFT 的逆.

我们写出 MATLAB 代码来实现这种想法. 粗略地讲, 我们想用 MATLAB 命令 fft 和 ifft 执行

$$\boldsymbol{F}_p^{-1}\sqrt{\frac{p}{n}}\boldsymbol{F}_n\boldsymbol{x},$$

这里

$$\boldsymbol{F}_p^{-1} = \sqrt{p}\cdot\mathtt{ifft},\quad \boldsymbol{F}_n = \frac{1}{\sqrt{n}}\cdot\mathtt{fft}.$$

把这两部分放在一起, 就对应以下的运算

$$\sqrt{p}\cdot\mathtt{ifft}_{[p]}\sqrt{\frac{p}{n}}\frac{1}{\sqrt{n}}\cdot\mathtt{fft}_{[n]} = \frac{p}{n}\cdot\mathtt{ifft}_{[p]}\cdot\mathtt{fft}_{[n]}. \tag{10.22}$$

当然, 只能对长度为 p 的向量用 $\boldsymbol{F}_p^{-1}$, 所以在求逆前, 我们需要把 n 个 Fourier 系数放置到长度为 p 的向量中. 短程序 **dftinterp.m** 执行这些步骤.

```
%Program 10.1 Fourier interpolation
%Interpolate n data points on [c,d] with trig function P(t)
%  and plot interpolant at p (>=n) evenly spaced points.
%Input: interval [c,d], data points x, even number of data
%  points n, even number p>=n
%Output: data points of interpolant xp
function xp=dftinterp(inter,x,n,p)
c=inter(1);d=inter(2);t=c+(d-c)*(0:n-1)/n; tp=c+(d-c)*(0:p-1)/p;
y=fft(x);                     % apply DFT
yp=zeros(p,1);                % yp will hold coefficients for ifft
yp(1:n/2+1)=y(1:n/2+1);       % move n frequencies from n to p
yp(p-n/2+2:p)=y(n/2+2:n);     % same for upper tier
xp=real(ifft(yp))*(p/n);      % invert fft to recover data
plot(t,x,'o',tp,xp)           % plot data points and interpolant
```

例如, 运行函数 **dftinterp ([0, 1], [-2.2, -2.8, -6.1, -3.9, 0.0, 1.1, -0.6, -1.1], 8, 100)** 并没有明显地用正弦和余弦, 就产生出绘在图 10-6 中的 $p = 100$ 个点. 依次对这段代码作一些说明. 目的是先应用 $\mathtt{fft}_{[n]}$ 再应用 $\mathtt{ifft}_{[p]}$ 然后再乘以 $\frac{p}{n}$. 对 $\boldsymbol{x}$ 的 n 个值应用 fft 后, 向量 y 的系数从 $P_n(t)$ 中的频率 n 转移到持有频率 p 的向量 yp, 这里 $p \geqslant n$. 在没有被 P_n 用到的频率 p 中有许多更高的频率, 这会导致位于 $\frac{n}{2}+2$ 到 $\frac{p}{2}+1$ 的那些高频率中的系数为零. 在 yp 的上半部分的分量值重复了下半部分的, 按照 (10.13) 以复共轭和逆序形式. 在用 ifft 命令对 DFT 求逆之后, 尽管理论上的结果是实数, 但是在计算上, 由于舍入, 可能会有小的虚部. 可通过使用 real 命令除去它.

特别简单的和有用的情形是 $c = 0, d = n$. 在整数插值节点 $s_j = j(j = 0, \cdots, n-1)$ 收集数据点 x_j, 用三角函数对点 (j, x_j) 插值:

$$P_n(s) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{k=1}^{n/2-1} \left(a_k \cos \frac{2k\pi}{n} s - b_k \sin \frac{2k\pi}{n} s \right) + \frac{a_{n/2}}{\sqrt{n}} \cos \pi s. \tag{10.23}$$

在第 11 章, 为了与通常的声音与图像数据的压缩算法相容, 我们将完全使用整数插值节点.

习题 10.2

1. 用 DFT 和推论 10.8, 对以下数据求三角插值函数:

(a)

t	x
0	0
$\frac{1}{4}$	1
$\frac{1}{2}$	0
$\frac{3}{4}$	–1

(b)

t	x
0	1
$\frac{1}{4}$	1
$\frac{1}{2}$	–1
$\frac{3}{4}$	–1

(c)

t	x
0	–1
$\frac{1}{4}$	1
$\frac{1}{2}$	–1
$\frac{3}{4}$	1

(d)

t	x
0	1
$\frac{1}{4}$	1
$\frac{1}{2}$	1
$\frac{3}{4}$	1

2. 利用 (10.23), 求以下数据的三角插值函数:

(a)

t	x
0	0
1	1
2	0
3	–1

(b)

t	x
0	1
1	1
2	–1
3	–1

(c)

t	x
0	1
1	2
2	4
3	1

(d)

t	x
0	1
1	0
2	1
3	0

3. 求以下数据的三角插值函数:

(a)

t	x
0	0
$\frac{1}{8}$	1
$\frac{1}{4}$	0
$\frac{3}{8}$	–1
$\frac{1}{2}$	0
$\frac{5}{8}$	1
$\frac{3}{4}$	0
$\frac{7}{8}$	–1

(b)

t	x
0	1
$\frac{1}{8}$	2
$\frac{1}{4}$	1
$\frac{3}{8}$	0
$\frac{1}{2}$	1
$\frac{5}{8}$	2
$\frac{3}{4}$	1
$\frac{7}{8}$	0

(c)

t	x
0	1
$\frac{1}{8}$	1
$\frac{1}{4}$	1
$\frac{3}{8}$	1
$\frac{1}{2}$	0
$\frac{5}{8}$	0
$\frac{3}{4}$	0
$\frac{7}{8}$	0

(d)

t	x
0	1
$\frac{1}{8}$	–1
$\frac{1}{4}$	1
$\frac{3}{8}$	–1
$\frac{1}{2}$	1
$\frac{5}{8}$	–1
$\frac{3}{4}$	1
$\frac{7}{8}$	–1

4. 求以下数据的三角插值函数:

(a)

t	x
0	0
1	1
2	0
3	–1
4	0
5	1
6	0
7	–1

(b)

t	x
0	1
1	2
2	1
3	0
4	1
5	2
6	1
7	0

(c)

t	x
0	1
1	0
2	1
3	0
4	1
5	0
6	1
7	0

(d)

t	x
0	–1
1	0
2	0
3	0
4	1
5	0
6	0
7	0

5. 在 n 是奇数的情形下，求插值函数 (10.19) 的形式.

计算机问题 10.2

1. 求下列数据的 8 阶三角插值函数 $P_8(t)$:

(a)

t	x
0	0
$\frac{1}{8}$	1
$\frac{1}{4}$	2
$\frac{3}{8}$	3
$\frac{1}{2}$	4
$\frac{5}{8}$	5
$\frac{3}{4}$	6
$\frac{7}{8}$	7

(b)

t	x
0	2
$\frac{1}{8}$	–1
$\frac{1}{4}$	0
$\frac{3}{8}$	1
$\frac{1}{2}$	1
$\frac{5}{8}$	3
$\frac{3}{4}$	–1
$\frac{7}{8}$	–1

(c)

t	x
0	3
1	1
2	4
3	2
4	3
5	1
6	4
7	2

(d)

t	x
1	1
2	–2
3	5
4	3
5	–2
6	–3
7	1
8	2

绘出数据点及 $P_8(t)$.

2. 求下列数据的 8 阶三角插值函数 $P_8(t)$:

(a)

t	x
0	6
$\frac{1}{8}$	5
$\frac{1}{4}$	4
$\frac{3}{8}$	3
$\frac{1}{2}$	2
$\frac{5}{8}$	1
$\frac{3}{4}$	0
$\frac{7}{8}$	–1

(b)

t	x
0	3
$\frac{1}{8}$	1
$\frac{1}{4}$	2
$\frac{3}{8}$	–1
$\frac{1}{2}$	–1
$\frac{5}{8}$	–2
$\frac{3}{4}$	3
$\frac{7}{8}$	0

(c)

t	x
0	1
2	2
4	4
6	–1
8	0
10	1
12	0
14	2

(d)

t	x
–7	2
–5	1
–3	0
–1	5
1	7
3	2
5	1
7	–4

绘出数据点及 $P_8(t)$.

3. 求 $f(t)=\mathrm{e}^t$ 在等距间隔点 $(\frac{j}{8}, f(j/8))$ $(j=0,\cdots,7)$ 的 $n=8$ 阶的三角插值函数.

4. 对 (a) $n=16$, (b) $n=32$, 绘出计算机问题 3 中在 $[0,1]$ 上的插值函数 $P_n(t)$, 以及数据点及 $f(t)=\mathrm{e}^t$.

5. 求 $f(t)=\ln t$ 在等距间隔点 $(1+\frac{j}{8}, f(1+\frac{j}{8}))(j=0,\cdots,7)$ 的 8 阶三角插值函数, 并绘出 $f(t)$、数据点和插值函数.
6. 对 (a) $n=16$, (b) $n=32$, 绘出计算机问题 5 中在 $[0,1]$ 上的插值函数 $P_n(t)$, 以及数据点及 $f(t)=\ln t$.

10.3 FFT 和信号处理

DFT 插值定理 10.6 只不过是 Fourier 变换的一种应用. 本节从更一般的观点来考虑插值, 这种观点将证明如何利用三角函数来求最小二乘近似. 这种想法形成了现代信号处理的基础. 它们将在第 11 章中第二次出现, 用于离散余弦变换.

10.3.1 正交性和插值

根据 $\boldsymbol{F}_n^{-1}=\bar{\boldsymbol{F}}_n^{\mathrm{T}}=\bar{\boldsymbol{F}}_n$, 这使得 $\boldsymbol{F}_n$ 是酉矩阵, 因此定理 10.6 中貌似简单的插值结果成为可能. 我们在第 4 章中遇到了这个定义的实形式, 如果 $\boldsymbol{U}^{-1}=\boldsymbol{U}^{\mathrm{T}}$, 我们就称矩阵 $\boldsymbol{U}$ 为正交阵. 现在我们研究正交矩阵的一种特殊形式, 它将马上转化为一种好的插值.

定理 10.9 (正交函数插值定理) 设 $f_0(t),\cdots,f_{n-1}(t)$ 是 t 的函数, 而且 $t_0,\cdots,t_{n-1}$ 是实数. 假设 $n\times n$ 矩阵

$$\boldsymbol{A}=\begin{bmatrix} f_0(t_0) & f_0(t_1) & \cdots & f_0(t_{n-1}) \\ f_1(t_0) & f_1(t_1) & \cdots & f_1(t_{n-1}) \\ \vdots & \vdots & & \vdots \\ f_{n-1}(t_0) & f_{n-1}(t_1) & \cdots & f_{n-1}(t_{n-1}) \end{bmatrix} \tag{10.24}$$

是实 $n\times n$ 正交矩阵. 如果 $\boldsymbol{y}=\boldsymbol{A}\boldsymbol{x}$, 那么函数

$$F(t)=\sum_{k=0}^{n-1} y_k f_k(t)$$

在 $(t_0,x_0),\cdots,(t_{n-1},x_{n-1})$ 处插值, 即 $F(t_j)=x_j, j=0,\cdots,n-1$.

证 $\boldsymbol{y}=\boldsymbol{A}\boldsymbol{x}$ 这个事实表明

$$\boldsymbol{x}=\boldsymbol{A}^{-1}\boldsymbol{y}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y},$$

由此得到

$$x_j=\sum_{k=0}^{n-1} a_{kj}y_k=\sum_{k=0}^{n-1} y_k f_k(t_j), \qquad j=0,\cdots,n-1.$$

这就完成了证明.

例 10.4 设 $[c,d]$ 是区间, n 是正的偶数. 证明对 $t_j = c + j(d-c)/n(j = 0,\cdots,n-1)$, 满足定理 10.9 的假设, 并且

$$
\begin{aligned}
f_0(t) &= \sqrt{\frac{1}{n}}, \\
f_1(t) &= \sqrt{\frac{2}{n}}\cos\frac{2\pi(t-c)}{d-c}, \\
f_2(t) &= \sqrt{\frac{2}{n}}\sin\frac{2\pi(t-c)}{d-c}, \\
f_3(t) &= \sqrt{\frac{2}{n}}\cos\frac{4\pi(t-c)}{d-c}, \\
f_4(t) &= \sqrt{\frac{2}{n}}\sin\frac{4\pi(t-c)}{d-c}, \\
&\vdots \\
f_{n-1}(t) &= \frac{1}{\sqrt{n}}\cos\frac{n\pi(t-c)}{d-c}.
\end{aligned}
$$

矩阵是

$$
\boldsymbol{A} = \sqrt{\frac{2}{n}}\begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\
1 & \cos\frac{2\pi}{n} & \cdots & \cos\frac{2\pi(n-1)}{n} \\
0 & \sin\frac{2\pi}{n} & \cdots & \sin\frac{2\pi(n-1)}{n} \\
\vdots & \vdots & & \vdots \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\cos\pi & \cdots & \frac{1}{\sqrt{2}}\cos(n-1)\pi
\end{bmatrix}. \tag{10.25}
$$

引理 10.10 表明, $\boldsymbol{A}$ 的行两两正交. ◀

引理 10.10 设 $n \geqslant 1, k, l$ 是整数, 那么

$$
\sum_{j=0}^{n-1}\cos\frac{2\pi jk}{n}\cos\frac{2\pi jl}{n} = \begin{cases} n, & \text{如果 } (k-l)/n \text{与} (k+l)/n \text{都是整数}, \\ \frac{n}{2}, & \text{如果} (k-l)/n \text{与} (k+l)/n \text{中恰有一个是整数}, \\ 0, & \text{如果两个都不是整数}, \end{cases}
$$

$$
\sum_{j=0}^{n-1}\cos\frac{2\pi jk}{n}\sin\frac{2\pi jl}{n} = 0,
$$

$$
\sum_{j=0}^{n-1}\sin\frac{2\pi jk}{n}\sin\frac{2\pi jl}{n} = \begin{cases} 0, & \text{如果} (k-l)/n \text{与} (k+l)/n \text{都是整数}, \\ \frac{n}{2}, & \text{如果} (k-l)/n \text{是整数但} (k+l)/n \text{不是}, \\ -\frac{n}{2}, & \text{如果} (k+l)/n \text{是整数但} (k-l)/n \text{不是}, \\ 0, & \text{如果两个都不是整数}. \end{cases}
$$

根据引理 10.1 可得到这个引理的证明. 见习题 5.

现在回到例 10.4, 设 $\boldsymbol{y}=\boldsymbol{Ax}$, 定理 10.9 马上给出点 (t_i, x_j) 的插值函数

$$\begin{aligned}F(t)=&\frac{1}{\sqrt{n}}y_0\\&+\sqrt{\frac{2}{n}}y_1\cos\frac{2\pi(t-c)}{d-c}+\sqrt{\frac{2}{n}}y_2\sin\frac{2\pi(t-c)}{d-c}\\&+\sqrt{\frac{2}{n}}y_3\cos\frac{4\pi(t-c)}{d-c}+\sqrt{\frac{2}{n}}y_4\sin\frac{4\pi(t-c)}{d-c}\\&\vdots\\&+\frac{1}{\sqrt{n}}y_{n-1}\cos\frac{n\pi(t-c)}{d-c},\end{aligned}\tag{10.26}$$

与 (10.19) 相一致.

例 10.5 使用例 10.4 的基函数, 对例 10.3 中的数据点 $\boldsymbol{x}=[-2.2,-2.8,-6.1,-3.9,0.0,1.1,-0.6,-1.1]$ 进行插值.

计算 8×8 矩阵 $\boldsymbol{A}$ 与 $\boldsymbol{x}$ 的乘积, 得到

$$\boldsymbol{Ax}=\sqrt{\frac{2}{8}}\begin{bmatrix}\frac{1}{\sqrt{2}}&\frac{1}{\sqrt{2}}&\frac{1}{\sqrt{2}}&\cdots&\frac{1}{\sqrt{2}}\\1&\cos 2\pi\frac{1}{8}&\cos 2\pi\frac{2}{8}&\cdots&\cos 2\pi\frac{7}{8}\\0&\sin 2\pi\frac{1}{8}&\sin 2\pi\frac{2}{8}&\cdots&\sin 2\pi\frac{7}{8}\\1&\cos 4\pi\frac{1}{8}&\cos 4\pi\frac{2}{8}&\cdots&\cos 4\pi\frac{7}{8}\\0&\sin 4\pi\frac{1}{8}&\sin 4\pi\frac{2}{8}&\cdots&\sin 4\pi\frac{7}{8}\\1&\cos 6\pi\frac{1}{8}&\cos 6\pi\frac{2}{8}&\cdots&\cos 6\pi\frac{7}{8}\\0&\sin 6\pi\frac{1}{8}&\sin 6\pi\frac{2}{8}&\cdots&\sin 6\pi\frac{7}{8}\\\frac{1}{\sqrt{2}}&\frac{1}{\sqrt{2}}\cos\pi&\frac{1}{\sqrt{2}}\cos 2\pi&\cdots&\frac{1}{\sqrt{2}}\cos 7\pi\end{bmatrix}\begin{bmatrix}-2.2\\-2.8\\-6.1\\-3.9\\0.0\\1.1\\-0.6\\-1.1\end{bmatrix}=\begin{bmatrix}-5.515\,4\\-1.488\,9\\-5.118\,8\\2.250\,0\\1.650\,0\\-0.711\,1\\0.381\,2\\-0.777\,8\end{bmatrix}.$$

公式 (10.26) 给出插值函数

$$\begin{aligned}P(t)=&-1.95-0.744\,5\cos 2\pi t-2.559\,4\sin 2\pi t+1.125\cos 4\pi t+0.825\sin 4\pi t\\&-0.355\,5\cos 6\pi t+0.190\,6\sin 6\pi t-0.275\,0\cos 8\pi t,\end{aligned}$$

这与例 10.3 相一致. ◀

10.3.2 用三角函数进行最小二乘拟合

推论 10.8 说明 DFT 如何用以下形式的三角函数

$$P_n(t)=\frac{a_0}{\sqrt{n}}+\frac{2}{\sqrt{n}}\sum_{k=1}^{\frac{n}{2}-1}(a_k\cos 2k\pi t-b_k\sin 2k\pi t)+\frac{a_{\frac{n}{2}}}{\sqrt{n}}\cos n\pi t,\tag{10.27}$$

使得对 $[0,1]$ 上 n 个等距间隔的数据点的插值变得容易.

亮点　正交性

在第 4 章, 为了求解由基函数表示的数据的最小二乘近似, 我们建立了正规方程 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\bar{\boldsymbol{x}}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$. 定理 10.9 的关键是求出使正规方程简单的特殊情形, 大大地简化了最小二乘的过程. 这就导出了极为有用的所谓正交函数的理论. 主要例子包括本章的 Fourier 变换和第 11 章的余弦变换.

注意项数 n 等于数据点的数目 (本章通常假定 n 是偶数). 数据点愈多, 帮助插值所加的余弦和正弦函数愈多.

如我们在第 3 章发现的, 当数据点的数目 n 较大时, 精确地拟合模型变得很少见. 事实上, 模型的一般应用是为了把事情简单化而忽略少数细节 (有损压缩). 在第 4 章讨论过的放弃插值的第二个原因是数据点本身被假设为不准确的, 因此插值函数据的严格执行是不合适的.

这两种情形启发我们用 (10.27) 这种类型的函数进行最小二乘拟合. 由于在模型中系数 a_k 和 b_k 是线性的, 我们可以用第 4 章所叙述的相同程序进行, 用正规方程求解最佳系数. 当这样尝试时, 我们发现一个意外的结果, 它正好把我们送回到 DFT.

回到定理 10.9, 设 n 表示数据点 x_j 的个数, 为了简单, 我们认为数据点出现在 $[0,1]$ 中等距的时间点 $t_j=\frac{j}{n}$ 上. 我们将引入正的偶数 m 来表示在最小二乘中使用的基函数的个数. 也就是我们将拟合前 m 个基函数 $f_0(t),\cdots,f_{m-1}(t)$. 用来拟合 n 个数据点的函数将是

$$P_m(t)=\sum_{k=0}^{m-1}c_kf_k(t), \tag{10.28}$$

这里 c_k 是需要确定的. 当 $m=n$ 时, 这个问题仍是插值. 当 $m<n$ 时, 问题已经改变成压缩问题. 在这种情形下, 我们要求以最小的平方误差用 P_m 来匹配数据点.

最小二乘问题是求系数 $c_0,\cdots,c_{m-1}$, 使得等式

$$\sum_{k=0}^{m-1}c_kf_k(t_j)=x_j$$

满足尽可能小的误差. 用矩阵表示,

$$\boldsymbol{A}_m^{\mathrm{T}}\boldsymbol{c}=\boldsymbol{x}, \tag{10.29}$$

这里 $\boldsymbol{A}_m$ 是 $\boldsymbol{A}$ 的前 m 行的矩阵. 在定理 10.9 的假设下, $\boldsymbol{A}_m^{\mathrm{T}}$ 有两两正交的列. 当对 c 建立正规方程

$$\boldsymbol{A}_m\boldsymbol{A}_m^{\mathrm{T}}\boldsymbol{c}=\boldsymbol{A}_m\boldsymbol{x}$$

时, $\boldsymbol{A}_m\boldsymbol{A}_m^{\mathrm{T}}$ 是单位矩阵, 因此最小二乘解

$$\boldsymbol{c} = \boldsymbol{A}_m\boldsymbol{x} \tag{10.30}$$

是容易计算的. 我们已证明了以下有用的结果, 它推广了定理 10.9.

定理 10.11 (正交函数最小二乘近似定理) 设 $m \leqslant n$ 是整数, 并且假设给定了数据 $(t_0, x_0), \cdots, (t_{n-1}, x_{n-1})$. 令 $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, 这里 $\boldsymbol{A}$ 是形式为 (10.24) 的正交矩阵. 那么关于基函数 $f_0(t), \cdots, f_{n-1}(t)$ 的插值多项式是

$$F_n(t) = \sum_{k=0}^{n-1} y_k f_k(t), \tag{10.31}$$

并且仅用函数 $f_0, \cdots, f_{m-1}$ 的最佳最小二乘近似是

$$F_m(t) = \sum_{k=0}^{m-1} y_k f_k(t). \tag{10.32}$$

这是一个极好而有用的事实. 它说明, 给定 n 个数据点, 要求用 $m < n$ 项拟合数据的最佳最小二乘三角函数, 那么用 n 项计算实际的内插, 而且只要保持需要的前 m 项就足够了. 换言之, 当项从最高频率降低时, 关于 $\boldsymbol{x}$ 的插值系数 $\boldsymbol{A}\boldsymbol{x}$ 尽可能适度地减少. 在 n 项展式中保持 m 个最低项保证了可能与 m 个最低频率项的最佳拟合. 这种性质反映了这些基函数的 "正交性".

前面定理 10.11 的论证容易被修改以证明更一般的事情. 我们说明了对前 m 个基函数如何求最小二乘解, 但是实际上, 阶并不相关; 我们可能已经确定了基函数的任一子集. 通过简单地去掉 (10.31) 中不包含在这个子集内的所有项求出了最小二乘解. 形式 (10.32) 是一种 "低通度" 滤波器, 即假定较低下标的函数具有较低 "频率"; 但是通过改变所保存的基函数子集, 我们可以简单地丢掉不想要的系数而保留任何有兴趣的频率.

现在我们回到三角多项式 (10.27) 并且说明如何对 n 个数据点进行 m 阶形式的拟合, 这里 $m < n$. 使用的基函数是例 10.4 中满足定理 10.9 的假设的函数. 定理 10.11 表明, 无论什么插值系数, 通过丢掉所有高于 m 阶的项就求得最佳 m 阶最小二乘近似. 我们已经获得以下应用.

推论 10.12 设 $[c, d]$ 是一个区间, 令 $m < n$ 是正的偶数, $\boldsymbol{x} = (x_0, \cdots, x_{n-1})$ 是一个 n 维实向量, 并且 $t_j = c + j(d-c)/n, j = 0, \cdots, n-1$. 设

$$\{a_0, a_1, b_1, a_2, b_2, \cdots, a_{\frac{n}{2}-1}, b_{\frac{n}{2}-1}, a_{\frac{n}{2}}\} = F_n\boldsymbol{x}$$

是关于 x 的插值系数, 使得

$$x_j = P_n(t_j) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{k=1}^{\frac{n}{2}-1} \left(a_k \cos\frac{2k\pi(t_j - c)}{d-c} - b_k \sin\frac{2k\pi(t_j - c)}{d-c} \right)$$

$$+\frac{a_{\frac{n}{2}}}{\sqrt{n}}\cos\frac{n\pi(t_j-c)}{d-c}, \qquad j=0,\cdots,n-1,$$

那么

$$P_m(t)=\frac{a_0}{\sqrt{n}}+\frac{2}{\sqrt{n}}\sum_{k=1}^{\frac{m}{2}-1}\left(a_k\cos\frac{2k\pi(t-c)}{d-c}-b_k\sin\frac{2k\pi(t-c)}{d-c}\right)+\frac{a_{\frac{m}{2}}}{\sqrt{n}}\cos\frac{n\pi(t-c)}{d-c}$$

是对数据 $(t_j, x_j), j=0,\cdots,n-1$ **的最佳** m **阶最小二乘拟合.**

体会定理 10.11 的能力的另一种方法是, 把它与我们前面对最小二乘模型已经使用过的单项式基函数进行比较. 拟合了点 $(0,3),(1,3),(2,5)$ 的最佳最小二乘抛物线是 $y=x^2-x+3$. 换言之, 模型 $y=a+bx+cx^2$ 关于这组数据的最佳系数是 $a=3, b=-1, c=1$ (在这种情形下, 因为平方误差是零, 它就是插值抛物线). 现在让我们对基函数的一组子集进行拟合：譬如说把模型变成 $y=a+bx$. 我们计算出最佳线性拟合是 $a=\frac{8}{3}, b=1$. 注意, 对于一次拟合的系数与对于二次拟合的对应系数没有明显的关系. 这一点恰是对于三角 (函数) 基函数不发生的. 对形式 (10.28) 的插值拟合或者最小二乘拟合明显包含了关于较低阶最小二乘拟合的全部信息.

因为利用了极简单的 DFT 来求解最小二乘, 所以写一个计算机程序执行这些步骤特别简单. 设 $m<n<p$ 是整数, 这里 n 是数据点个数, m 是最小二乘三角模型的阶数, p 控制最佳模型的曲线图的分辨率. 可以把最小二乘理解成把 n 阶插值式的最高频率项 “过滤掉”, 仅仅留下最低的 m 个频率项. 这解释了以下 MATLAB 函数的名称.

```
% Program 10.2 Least squares trigonometric fit
% Least squares fit of n data points on [0,1] with trig function
%    where 2 <=m <=n. Plot best fit at p (>=n) points.
% Input: interval [c,d], data points x, even number m,
%    even number of data points n, even number p>=n
% Output: filtered points xp
function xp=dftfilter(inter,x,m,n,p)
c=inter(1); d=inter(2);
t=c+(d-c)*(0:n-1)/n;              % time points for data (n)
tp=c+(d-c)*(0:p-1)/p              % time points for interpolant (p)
y=fft(x);                         % compute interpolation coefficients
yp=zeros(p,1);                    % will hold coefficients for ifft
yp(1:m/2)=y(1:m/2);               % keep only first m frequencies
yp(m/2+1)=real(y(m/2+1));         % since m is even, keep cos term only
if(m<n)                           % unless at the maximum frequency,
  yp(p-m/2+1)=yp(m/2+1);          %    add complex conjugate to
end                               %    corresponding place in upper tier
yp(p-m/2+2:p)=y(n-m/2+2:n);       % more conjugates for upper tier
xp=real(ifft(yp))*(p/n);          % invert fft to recover data
plot(t,x,'o',tp,xp)               % plot data and least square approx
```

例 10.6　用 4 阶和 6 阶最小二乘三角函数, 拟合例 10.3 中的温度数据.

推论 10.12 的关键点是我们可以仅用 F_n 对数据点插值，然后丢掉一些项以得到较低阶的最小二乘拟合. 例 10.3 得到的结果是

$$P_8(t) = -1.95 - 0.744\ 5\cos 2\pi t - 2.559\ 4\sin 2\pi t + 1.125\cos 4\pi t + 0.825\sin 4\pi t \\ - 0.355\ 5\cos 6\pi t + 0.190\ 6\sin 6\pi t - 0.275\ 0\cos 8\pi t. \tag{10.33}$$

因此，4 阶和 6 阶的最小二乘模型是

$$P_4(t) = -1.95 - 0.744\ 5\cos 2\pi t - 2.559\ 4\sin 2\pi t + 1.125\cos 4\pi t$$

$$P_6(t) = -1.95 - 0.744\ 5\cos 2\pi t - 2.559\ 4\sin 2\pi t \\ +1.125\cos 4\pi t - 0.825\sin 4\pi t - 0.355\ 5\cos 6\pi t.$$

图 10-7 表示这两种最小二乘模型，分别对 $m = 4$ 及 $m = 6$，它们由

```
dftfilter([0,1],[-2.2,-2.8,-6.1,-3.9,0.0,1.1,-0.6,-1.1],m,8,200)
```

产生. $m = 4$ 的拟合与在例 4.6 中对基函数 $1, \cos 2\pi t, \sin 2\pi t, \cos 4\pi t$ 执行并在图 4-5b 中绘出的显式最小二乘拟合相匹配. ◀

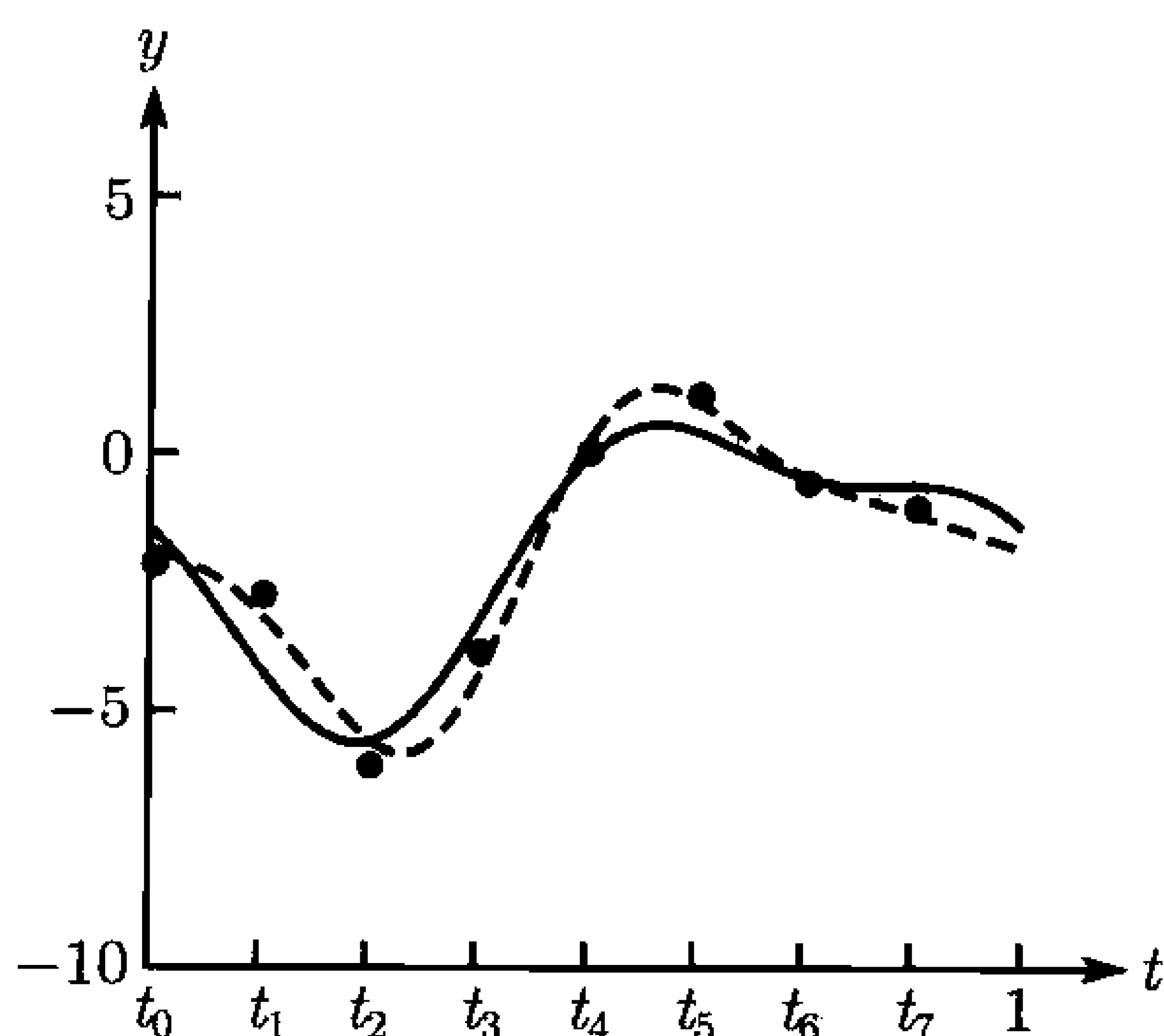


图 10-7 例 10.6 的最小二乘三角拟合. 表示 $m = 4$(实曲线) 和 $m = 6$(虚曲线) 的拟合. 输入向量 x 是 $[-2.2, -2.8, -6.1, -3.9, 0.0, 1.1, -0.6, -1.1]^{\mathrm{T}}$. $m = 8$ 的拟合是三角插值，如图 10-6 所示

程序 `dftfilter.m` 可以变得更有效. 它计算 n 阶插值，然后把 $n - m$ 个系数略去不计. 当然，Fourier 矩阵 F_n 表明，如果只想知道 n 个数据点的前 m 个 Fourier 系数，我们可以仅用 F_n 的前 m 行与 x 相乘并把它留在那里. 换言之，用 $m \times n$ 子矩阵代替 $n \times n$ 矩阵 F_n 就足够了. 一种改进形式的 `dftfilter.m` 可能利用这个事实.

10.3.3 声音、噪声和滤波

10.3.2 节的 `dftfilter.m` 代码是对应用广泛的数字信号处理的一种介绍. 我们用 Fourier 变换把信号 $\{x_0, \cdots, x_{n-1}\}$ 从“时域”转变到较易操作的“频域”. 当

完成想要的改变时, 我们通过逆 FFT 把信号送回时域.

如果 x 表示音频信号, 那么基于听觉系统的构造方法, 这是有益的. 人耳包括对频率产生感应的结构, 所以在频域中构造块恰恰是意义深远的. 我们将通过引入某些音频和信号处理的基本概念和一些方便的 MATLAB 命令来说明这一点.

音频信号由一组用时间指示的实数组成. 每一个实数表示一种声音强度. 当一种音频信号回放时, 发音者的头振动使得振幅与信号匹配, 造成周围空气以相同的频率振动. 当声波传播到你耳朵时, 你感觉到声音.

我们选取 MATLAB 提供的 Handel 的哈利路亚合唱曲 (Hallelujah Chorus) 的最初 9 秒钟的音频信号为作样本. 图 10-8 中的曲线表示文件的最初 $2^8 = 256$ 个值, 它们是由声音强度组成. 音乐的抽样速度是 $2^{13} = 8\ 192$ Hz, 意即相当于等距间隔每秒 2^{13} 的速度的强度. 为了获取信号, 输入

```
>> load handel
```

把变量 $\boldsymbol{F_s}$ 和 $\boldsymbol{y}$ 放入工作空间. 前一个变量是抽样速度 $F_s = 8\ 192$. 变量 y 是包含声音信号且长度为 73 113 的向量. MATLAB 命令

```
>> sound(y,Fs)
```

如果可行的话, 以正确的抽样速度 F_s 在你的计算机扬声器上播放信号.

可以用哈利路亚合唱曲的数据执行推论 10.12 中的滤波. 用 `dftfiter.m` 处理信号的最初 $n = 256$ 个样本以及 $m = 64$ 及 $m = 32$ 个基函数, 结果如图 10-8 中的实线所示. 读者可能想要研究其他音频文件的滤波.

一种普通的音频文件格式是 `.wav` 格式. 立体声的 `.wav` 文件执行两对来自两个不同扬声器的信号. 例如, 用 MATLAB 命令

```
>> [y,Fs]=wavread('castanets')
```

将从文件 `castanets.wav` 取出立体声信号并把它放入 MATLAB 作为一个 $n \times 2$ 矩阵 $\boldsymbol{y}$, 每一列都是分离的声音信号 (文件 `castanets.wav` 是一个普通的音频测试文件, 通过网上搜查容易找到它). MATLAB命令 `wavwrite` 把过程反转过来, 由简单的声音信号产生一个`.wav` 文件.

滤波用于两种方式. 它能够用来把原来的声波与一个较简单的函数尽可能接近地匹配. 这是一种压缩的形式. 不是用 256 个数字存储波, 我们可以恰好存储最低的 m 个频率的分量, 然后, 当需要时用推论 10.12 来重新构造波. 在图 10-8a 中, 我们在原来 256 的地方用了 $m = 64$ 个实数, 这是一个 $4:1$ 的压缩比. 注意这个压缩是有损耗的, 在压缩中原来的波没有准确地重制.

滤波的第二个主要应用是清理杂音. 给定一个音乐文件, 其中音乐或演讲话音搀杂了高频杂音 (或嘶嘶声), 消除高频项可能对提高声音质量是重要的. 当然, 所

谓的低通度滤波或许并不好用：所要声音的高频部分，可能在泛音区但听者感觉不明显，也可能被除去．滤波的话题是关于信号处理的大量文献中的一部分，为了进一步学习，读者可参阅 [9]．在实例检验 10 中，我们研究 Wiener 滤波器，这是一类应用广泛的滤波器．

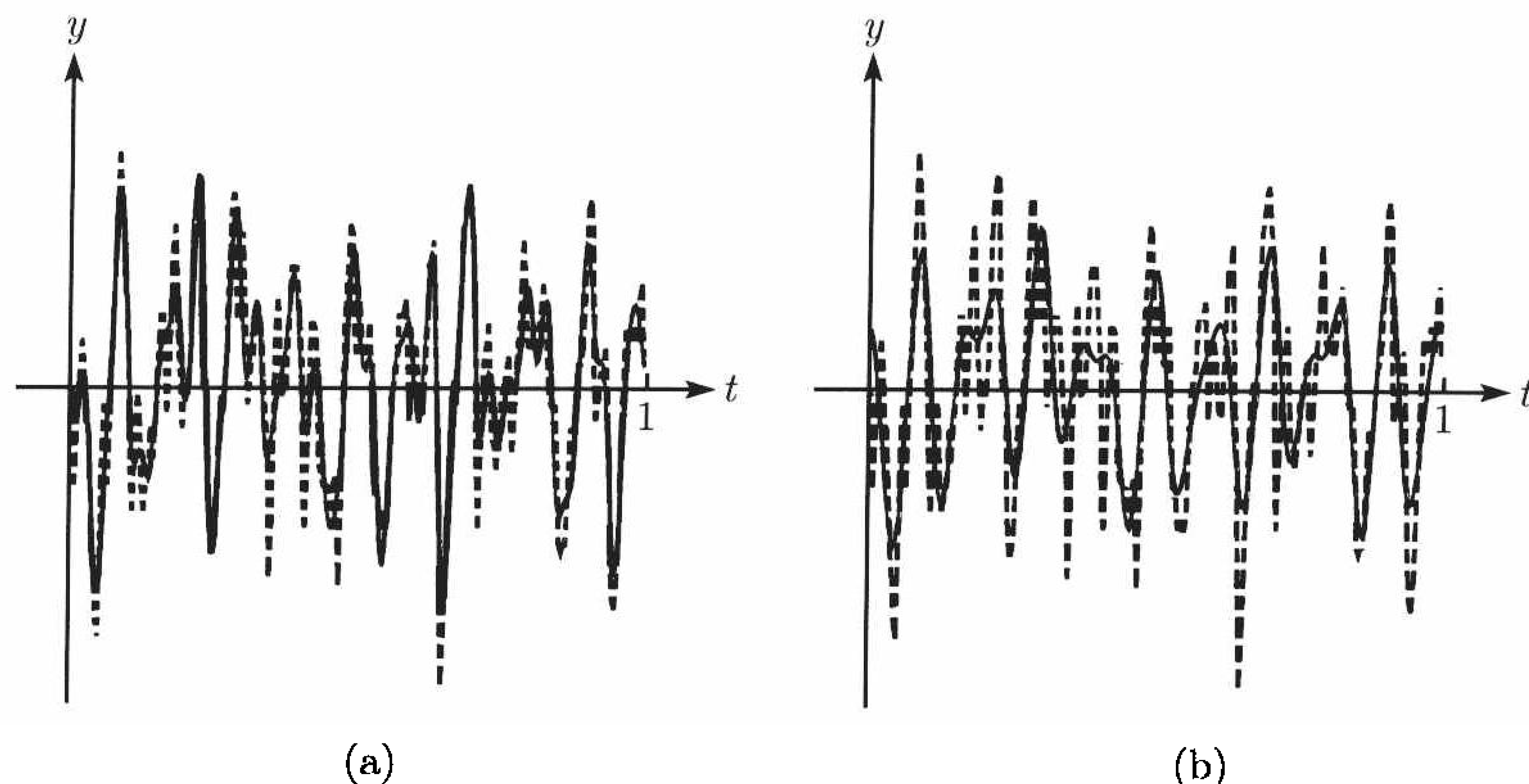


图 10-8 声音曲线以及滤波形式哈利路亚合唱曲的最初 $\frac{1}{32}$ 秒 (虚线的 256 个点) 以及滤波形式 (实线)：(a) 用 64 个基函数，4 : 1 的压缩比；(b) 用 32 个基函数，8 : 1 的压缩比

亮点 压缩

滤波是一种有损耗的压缩形式．在音频信号的情形下，目标是减少需要储存或传输声音的数据数量，而又不损害这种信号所表示的音乐效果或演讲信息．这在频域中完成得最好，这意味着应用 DFT 处理频率分量，然后求 DFT 的逆．

习题 10.3

1. 取基函数 1 和 $\cos 2\pi t$，对习题 10.2.1 中的数据求最佳 2 阶最小二乘近似．
2. 取基函数 $1, \cos 2\pi t$ 和 $\sin 2\pi t$，对习题 10.2.1 中的数据求最佳 3 阶最小二乘近似．
3. 取基函数 $1, \cos 2\pi t, \sin 2\pi t$ 和 $\cos 4\pi t$，对习题 10.2.3 中的数据求最佳 4 阶最小二乘近似．
4. 取基函数 $1, \cos \frac{\pi}{4}t, \sin \frac{\pi}{4}t$ 和 $\cos \frac{\pi t}{2}$，对习题 10.2.4 中的数据求最佳 4 阶最小二乘近似．
5. 证明引理 10.10 [提示：把 $\cos \frac{2\pi jk}{n}$ 表示为 $(e^{2\pi ijk/n} + e^{-2\pi ijk/n})/2$，并把每一项写成用 $\omega = e^{-2\pi i/n}$ 来表示，所以能够用引理 10.1]．

计算机问题 10.3

1. 对以下数据点求 $m = 2$ 和 $m = 4$ 阶的最小二乘三角逼近函数：

(a)

t	y
0	3
$\frac{1}{4}$	0
$\frac{1}{2}$	−3
$\frac{3}{4}$	0

(b)

t	y
0	2
$\frac{1}{4}$	0
$\frac{1}{2}$	5
$\frac{3}{4}$	1

(c)

t	y
0	5
1	2
2	6
3	1

(d)

t	y
0	−1
2	1
3	4
4	3
5	3
6	2

使用 `dftfilter.m`, 如在图 10-7 中那样, 绘出数据点及逼近函数.

2. 对以下数据点求 4 阶, 6 阶和 8 阶的最小二乘三角逼近函数:

(a)

t	y
0	3
$\frac{1}{8}$	0
$\frac{1}{4}$	−3
$\frac{3}{8}$	0
$\frac{1}{2}$	3
$\frac{5}{8}$	0
$\frac{3}{4}$	−6
$\frac{7}{8}$	0

(b)

t	y
0	1
$\frac{1}{8}$	0
$\frac{1}{4}$	−2
$\frac{3}{8}$	1
$\frac{1}{2}$	3
$\frac{5}{8}$	0
$\frac{3}{4}$	−2
$\frac{7}{8}$	1

(c)

t	y
0	1
$\frac{1}{8}$	2
$\frac{1}{4}$	3
$\frac{3}{8}$	1
$\frac{1}{2}$	−1
$\frac{5}{8}$	−1
$\frac{3}{4}$	−3
$\frac{7}{8}$	0

(d)

t	y
0	4.2
$\frac{1}{8}$	5.0
$\frac{1}{4}$	3.8
$\frac{3}{8}$	1.6
$\frac{1}{2}$	−2.0
$\frac{5}{8}$	−1.4
$\frac{3}{4}$	0.0
$\frac{7}{8}$	1.0

如在图 10-7 中那样, 绘出数据点及逼近函数.

3. 用 MATLAB 的 `handel` 声音文件, 绘出 $m = \frac{n}{2}, \frac{n}{4}, \frac{n}{8}$ 阶的最小二乘三角逼近函数以及包含最初 2^{14} 个声音强度值的向量 x (这覆盖了大约两秒的音频. 对 $p = n$ 用 MATLAB 代码 `dftfilter`. 作出 3 个分开的曲线图). 使用 MATLAB 的 `sound` 命令把原来的与其近似进行比较. 已经失去了什么?

4. 从适当的网站下载 `castanets.wav`, 并形成包括最初 2^{14} 个样本时间的信号向量. 对每一个立体声波段分别执行计算机问题 3 中的步骤.

5. 从报纸或网站采集连续 24 小时温度测量记录. 绘出数据点以及 (a) 三角插值函数、(b) $m = 6$ 和 (c) $m = 12$ 阶的最小二乘逼近函数.

实例检验 10 WIENER 滤波器

设 $\boldsymbol{c}$ 是无噪声的音频信号, 并且把相同长度的向量 $\boldsymbol{r}$ 加到 $\boldsymbol{c}$ 上, 结果得到的信号 $\boldsymbol{x} = \boldsymbol{c}+\boldsymbol{r}$ 是否有噪声? 如果 $\boldsymbol{r} = \boldsymbol{c}$, 我们可能不认为 $\boldsymbol{r}$ 是噪声, 这是因为结果可能是较大的音频信号, 但仍是无噪声的 $\boldsymbol{c}$ 的形式. 根据定义, 噪声是与信号不相关的. 换言之, 如果 $\boldsymbol{r}$ 是噪声, 内积 $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{r}$ 的期望值是零. 下面我们将用到这种不相关性.

在典型的应用中, 我们会面临噪声信号 $\boldsymbol{x}$ 而要求去求 $\boldsymbol{c}$. 信号 $\boldsymbol{c}$ 可能是重要的系统变量的值, 在噪声环境中受到监控. 或者如在下面的例子里, $\boldsymbol{c}$ 可能是我们想引出噪声的音频样本. 在 20 世纪中叶, Norbert Wiener 为了从 $\boldsymbol{x}$ 消去噪声, 建议在最小二乘误差意义上寻找最佳滤波. 他建议求一个实对角矩阵 $\boldsymbol{\Phi}$ 使得

$$F^{-1}\boldsymbol{\Phi}F\boldsymbol{x} - \boldsymbol{c}$$

的 Euclid 范数尽可能地小, 这里 F 表示离散 Fourier 变换. 这个想法是通过 Fourier 变换来净化信号 $\boldsymbol{x}$, 用 $\boldsymbol{\Phi}$ 去乘频率分量, 然后对 Fourier 变换取逆. 这称为在频域内滤波, 因为我们在改变 $\boldsymbol{x}$ 的 Fourier 变换的形式而不是 $\boldsymbol{x}$ 本身.

为了求最佳对角矩阵 $\boldsymbol{\Phi}$, 注意

$$\begin{aligned}\|F^{-1}\boldsymbol{\Phi}F\boldsymbol{x}-\boldsymbol{c}\|_2 &= \|\boldsymbol{\Phi}F\boldsymbol{x}-F\boldsymbol{c}\|_2 = \|\boldsymbol{\Phi}(F(\boldsymbol{c}+\boldsymbol{r})-F\boldsymbol{c}\|_2\\ &= \|(\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{C}+\boldsymbol{\Phi}\boldsymbol{R}\|_2,\end{aligned} \tag{10.34}$$

这里我们令 $\boldsymbol{C}=F\boldsymbol{c}$ 及 $\boldsymbol{R}=F\boldsymbol{r}$ 是 Fourier 变换. 还注意到噪声的定义意味着

$$\bar{\boldsymbol{C}}^{\mathrm{T}}\boldsymbol{R}=\overline{F\boldsymbol{c}}^{\mathrm{T}}F\boldsymbol{r}=\boldsymbol{c}^{\mathrm{T}}\bar{F}^{\mathrm{T}}F\boldsymbol{r}=\boldsymbol{c}^{\mathrm{T}}\boldsymbol{r}=0.$$

我们将依据它而忽略掉范数中的交叉项, 所以长度的平方减小到

$$\begin{aligned}\left(\overline{(\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{C}+\boldsymbol{\Phi}\boldsymbol{R}}\right)^{\mathrm{T}}((\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{C}+\boldsymbol{\Phi}\boldsymbol{R}) &= (\bar{\boldsymbol{C}}^{\mathrm{T}}(\boldsymbol{\Phi}-\boldsymbol{I})+\bar{\boldsymbol{R}}^{\mathrm{T}}\boldsymbol{\Phi})((\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{C}+\boldsymbol{\Phi}\boldsymbol{R})\\ &\approx \bar{\boldsymbol{C}}^{\mathrm{T}}(\boldsymbol{\Phi}-\boldsymbol{I})^2\boldsymbol{C}+\bar{\boldsymbol{R}}^{\mathrm{T}}\boldsymbol{\Phi}^2\boldsymbol{R}\\ &= \sum_{i=1}^{n}(\varphi_i-1)^2|C_i|^2+\varphi_i^2|R_i|^2.\end{aligned} \tag{10.35}$$

为了求使这个表达式取极小的对角元素 φ_i, 分别对每一个 φ_i 求导得到

$$2(\varphi_i-1)|C_i|^2+2\varphi_i|R_i|^2=0$$

(对每一个 i), 或者解出 φ_i:

$$\varphi_i=\frac{|C_i|^2}{|C_i|^2+|R_i|^2}. \tag{10.36}$$

这个公式给出了对角矩阵 $\boldsymbol{\Phi}$ 的元素的 Wiener 值, 使滤波形式 $F^{-1}\boldsymbol{\Phi}F\boldsymbol{x}$ 和无噪声信号 $\boldsymbol{c}$ 的差极小, 仅有的问题是在典型情形下, 我们并不知道 $\boldsymbol{C}$ 或 $\boldsymbol{R}$, 因此必须作某些近似来用这个公式.

我们的任务是研究把近似放在一起的方法. 令 $\boldsymbol{X}=F\boldsymbol{x}$ 是 Fourier 变换. 再次应用信号和噪声的不相关性进行近似

$$|X_i|^2\approx|C_i|^2+|R_i|^2.$$

然后我们可以把最优选择写为

$$\varphi_i\approx\frac{|X_i|^2-|R_i|^2}{|X_i|^2}, \tag{10.37}$$

并且利用我们所能了解的噪声水平. 例如, 如果噪声是不相关的 Gauss 噪声 (对每一个无噪声信号样本, 独立地加上正规随机数模拟得到), 那么可以用常数 $(p\sigma)^2$ 代替 (10.37) 中的 $|R_i|^2$, 这里 σ 是噪声的标准差, 而 p 是有待选取的接近 1 的参数. 注意到

$$\sum_{i=1}^{n}|R_i|^2=\bar{\boldsymbol{R}}^{\mathrm{T}}\boldsymbol{R}=\boldsymbol{r}^{\mathrm{T}}\bar{F}^{\mathrm{T}}F\boldsymbol{r}=\boldsymbol{r}^{\mathrm{T}}\boldsymbol{r}=\sum_{i=1}^{n}r_i^2.$$

在下面的代码中, 我们对 Handel 信号加 50% 噪声, 并且用 $p=1.3$ 标准差来近似 R_i:

```
load handel                               % y is clean signal
c=y(1:40000);                             % work with first 40K samples
p=1.3;                                    % parameter for cutoff
noise=std(c)*.50;                         % 50 percent noise
n=length(c);                              % n is length of signal
r=noise*randn(n,1);                       % pure noise
x=c+r;                                    % noisy signal
fx=fft(x);sfx=conj(fx).*fx;               % take fft of signal, and
sfcapprox=max(sfx-n*(p*noise)^2,0);       % apply cutoff
phi=sfcapprox./sfx;                       % define phi as derived
xout=real(ifft(phi.*fx));                 % invert the fft
% then compare sound(x) and sound(xout)
```

建议习题

1. 运行上述代码产生滤波的信号 yf, 并且用 MATLAB 的 **sound** 命令来比较输入信号和输出信号.
2. 通过与无噪声信号 (yc) 比较, 计算输入 (ys)和输出 (yf)的平均平方误差 (MSE).
3. 对 50% 噪声, 求参数 p 的最佳值. 把极小化 MSE 的值与人耳听到的最佳值进行比较.
4. 把噪声水平改变成 10%, 25%, 100%, 200%, 并且重复第 3 步. 总结你的结论.
5. 对在 10.2 节中叙述的带有低通度滤波的 Wiener 滤波器设计一种公平合理的比较, 并且执行这种比较.
6. 下载你选取的 .wav 文件, 增加噪声, 并且执行上述步骤.

软件和进一步阅读

关于离散 Fourier 变换的进一步阅读资料包括 [3, 1, 2]. Cooley 和 Tukey 最初的突破见 [5]. 在现代信号处理中, 作为核心地位的快速 Fourier 变换在计算上一直在持续改进, 见 [11, 10, 4]. FFT 本身就是一种重要的算法, 而且因为它可以有效实现, 还用作其他算法中的构造块. 例如它被 MATLAB 用于计算在第 11 章定义的离散余弦变换. Cooley 和 Tukey 使用的分治策略随后成功地用到了许多其他的计算问题中.

MATLAB 的 **fft** 命令基于 20 世纪 90 年代在 MIT 发展起来的 "西方最快 Fourier 变换"(FFTW) [7] 在大小 n 不是 2 的幂的情形下, 利用 n 的素数因子, 程序把问题分解成较小的针对特定大小优化的 "代码". 关于 FFTW 的更多信息, 包括可下载代码, 在 http:/www.fftw.org 可以得到. IMSL 提供了前向变换 FFTCF 以及逆变换 FFTCB, 它们基于 Netlib 的 FFTPACK[8], 一种为在并行实现中使用而优化的快速 Fourier 变换的 Fortran 子程序包.

第 11 章　压　　缩

世界各地愈来愈快的信息传递依赖于数据表示的精巧方法, 正交变换使之成为可能. JPEG 格式的图像表示基于本章所介绍的离散余弦变换 (DCT). MPEG-1 和 MPEG-2 格式的电视数据和视频格式, 以及 H.263 格式的视频电话都是基于 DCT, 但更多地强调了时间维度上的压缩.

声音文件能被压缩成各种不同的格式, 包括 MP3、高级音频编码 (Advanced Audio Coding, 用于 Apple 公司的 iTunes 和 XM 卫星广播)、WMA(微软音频媒体) 和其他一些最新流行的格式. 这些格式有一个共同点, 就是它们的核心压缩都是基于改进的离散余弦变换 (Modified Discrete Cosine Transform, MDCT).

实例检验　11.4.2 节后的实例检验 11 把 MDCT 开发成一个简单有效的算法来压缩音频文件.

在第 4 章和第 10 章, 我们已经知道正交性在表示和压缩数据方面的用处. 第 11 章将介绍只用实数算术运算就能计算的 Fourier 变换的变形 (DCT). 它是当前我们用来压缩声音和图像文件的一种方法.

Fourier 变换可以表示成一个复的酉矩阵, 而它的简单正是由于它的正交性. DCT 方法能表示成一个实的正交矩阵, 正是它的正交性使得它容易被使用及进行逆变换. 它与离散 Fourier 变换充分相似以致存在 DCT 的快速形式, 这类似于快速 Fourier 变换 (FFT).

本章, 我们将详细讲解 DCT 的一些基本性质, 并考察这些性质和有效压缩格式之间的联系. 例如, 著名的 JPEG 格式把二维的 DCT 应用到 8×8 像素块的图像, 并且保存了使用 Huffman 编码的结果. 关于 JPEG 压缩的细节将在 11.2 节和 11.3 节作为一个案例来介绍.

一个称为 MDCT 的改进离散余弦变换方法是很多现代音频压缩格式的基础. 改进的离散余弦变换是当前声音文件压缩的黄金标准. 我们将介绍 MDCT 并且考察它在编码与解码中的应用, 它提供了 MP3 和 AAC 等文件格式的核心技术.

11.1　离散余弦变换

本节介绍离散余弦变换 (DCT). 这种变换使用余弦函数作为基函数对数据进行插值, 并且只涉及实数算术运算. 它的正交性特征使得最小二乘逼近像离散 Fourier

变换一样简单.

11.1.1　一维离散余弦变换

设 n 为正整数, 阶数为 n 的一维 DCT 由一个 $n \times n$ 的矩阵 $\boldsymbol{C}$ 表示, $\boldsymbol{C}$ 的元素为

$$C_{ij} = \sqrt{\frac{2}{n}} \begin{cases} 1/\sqrt{2}, & i = 0, j = 0, \cdots, n-1, \\ \cos \frac{i(j+\frac{1}{2})\pi}{n}, & i > 0, j = 0, \cdots, n-1, \end{cases} \tag{11.1}$$

或者

$$C = \sqrt{\frac{2}{n}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(2n-1)\pi}{2n} \\ \cos \frac{2\pi}{2n} & \cos \frac{6\pi}{2n} & \cdots & \cos \frac{2(2n-1)\pi}{2n} \\ \vdots & \vdots & & \vdots \\ \cos \frac{(n-1)\pi}{2n} & \cos \frac{(n-1)3\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \end{pmatrix}. \tag{11.2}$$

对于二维图像, 我们约定下标从零开始 (而不是从 1 开始). 这样的记号使得我们列举出元素变得更容易, 正如在 (11.1) 中一样.

本章, $n \times n$ 的矩阵下标将从 0 变化到 $n-1$. 为简单起见, 我们在下面的讨论中只处理 n 为偶数的情况.

定义 11.1(DCT)　如果

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}, \tag{11.3}$$

其中 $\boldsymbol{C}$ 如 (11.2) 所示, 则称 $\boldsymbol{y} = [y_0, \cdots, y_{n-1}]^{\mathrm{T}}$ 是 $\boldsymbol{x} = [x_0, \cdots, x_{n-1}]^{\mathrm{T}}$ 的离散余弦变换 (DCT).

注意到 $\boldsymbol{C}$ 是一个实的正交矩阵, 这意味着它的转置也就是它的逆为

$$\boldsymbol{C}^{-1} = \boldsymbol{C}^{\mathrm{T}} = \sqrt{\frac{2}{n}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \cos \frac{\pi}{2n} & \cdots & \cos \frac{(2n-1)\pi}{2n} \\ \frac{1}{\sqrt{2}} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(n-1)3\pi}{2n} \\ \vdots & \vdots & & \vdots \\ \frac{1}{\sqrt{2}} & \cos \frac{(2n-1)\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \end{pmatrix}. \tag{11.4}$$

正交矩阵的行是两两正交的单位向量. $\boldsymbol{C}$ 的正交性由以下事实得来: $\boldsymbol{C}^{\mathrm{T}}$ 的列是 $n \times n$ 实对称矩阵

$$\begin{pmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{pmatrix} \tag{11.5}$$

的单位特征向量. 习题 6 要求读者验证这一事实.

$\boldsymbol{C}$ 是实正交阵这一事实使得 DCT 有实际用途. 把正交函数插值定理 10.9 应用到矩阵 $\boldsymbol{C}$ 便得到定理 11.2.

定理 11.2(DCT 插值定理) 设 $\boldsymbol{x}=[x_0,\cdots,x_{n-1}]^{\mathrm{T}}$ 是一个 n 维实向量, 定义 $\boldsymbol{y}=[y_0,\cdots,y_{n-1}]^{\mathrm{T}}=\boldsymbol{Cx}$, 其中 $\boldsymbol{C}$ 是 DCT 矩阵. 则实函数

$$P_n(t)=\frac{1}{\sqrt{n}}y_0+\frac{\sqrt{2}}{\sqrt{n}}\sum_{k=1}^{n-1}y_k\cos\frac{k(2t+1)\pi}{2n}$$

满足 $P_n(t)=x_j$, 其中 $j=0,\cdots,n-1$.

证 由定理 10.9 直接可得.

定理 11.2 表明了 $n\times n$ 矩阵 $\boldsymbol{C}$ 可把 n 个数据点变换为 n 个插值系数. 像离散 Fourier 变换一样, DCT 给出了一个三角插值函数的系数. 不像 DFT, DCT 只使用了余弦函数项并且完全由实的算术运算所确定.

例 11.1 用 DCT 对点 (0, 1), (1, 0), (2, −1), (3, 0) 进行插值.

使用基本的三角函数运算, 可以把 4×4 的 DCT 矩阵看成

$$\boldsymbol{C}=\frac{1}{\sqrt{2}}\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & \cos\frac{5\pi}{8} & \cos\frac{7\pi}{8} \\ \cos\frac{2\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{10\pi}{8} & \cos\frac{14\pi}{8} \\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8} & \cos\frac{15\pi}{8} & \cos\frac{21\pi}{8} \end{pmatrix}=a\begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix}, \tag{11.6}$$

其中

$$a=\cos\frac{\pi}{4}=\frac{1}{\sqrt{2}},\quad b=\cos\frac{\pi}{8}=\frac{\sqrt{2+\sqrt{2}}}{2},\quad c=\cos\frac{3\pi}{8}=\frac{\sqrt{2-\sqrt{2}}}{2}. \tag{11.7}$$

4 阶的 DCT 矩阵乘以数据 $\boldsymbol{x}=(1,0,-1,0)^{\mathrm{T}}$ 为

$$a\begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ a(c+b) \\ 2a^2 \\ a(c-b) \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ \frac{\sqrt{2-\sqrt{2}}+\sqrt{2+\sqrt{2}}}{2\sqrt{2}} \\ 1 \\ \frac{\sqrt{2-\sqrt{2}}-\sqrt{2+\sqrt{2}}}{2\sqrt{2}} \end{pmatrix} \approx \begin{pmatrix} 0.000\ 0 \\ 0.923\ 9 \\ 1.000\ 0 \\ -0.382\ 7 \end{pmatrix}.$$

根据定理 11.2, 其中 $n=4$, 函数

$$P_4(t) = \frac{1}{\sqrt{2}}\left[0.923\ 9\cos\frac{(2t+1)\pi}{8} + \cos\frac{2(2t+1)\pi}{8} - 0.382\ 7\cos\frac{3(2t+1)\pi}{8}\right] \tag{11.8}$$

对这 4 个数据点进行了插值. 函数 $P_4(t)$ 如图 11-1 的实线所示. ◀

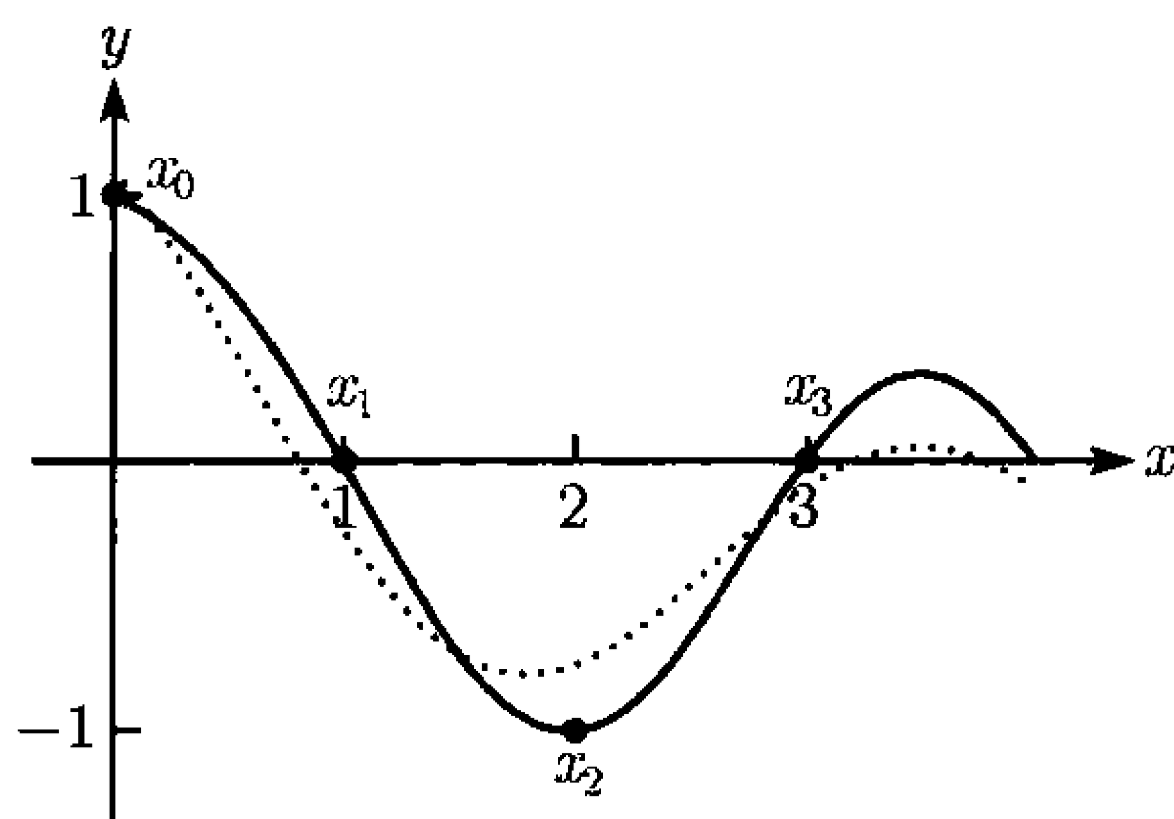


图 11-1 DCT 插值和最小二乘逼近. 数据点是 (j, x_j), 其中 $x = (1,0,-1,0)^{\mathrm{T}}$. (11.8) 式中的 DCT 插值函数 $P_4(t)$ 如实线所示, (11.9) 式中的最小二乘 DCT 逼近函数 $P_3(t)$ 如点虚线所示

11.1.2 DCT 和最小二乘逼近

正像由定理 10.9 马上就能得到 DCT 插值定理 11.2, 定理 10.11 的最小二乘结果表明了, 如何只利用部分基函数寻找数据的 DCT 最小二乘逼近. 由于基函数的正交性, 我们把高频项丢掉就完成这一步.

定理 11.3(DCT 最小二乘逼近定理) *设 $\boldsymbol{x} = [x_0, \cdots, x_{n-1}]^{\mathrm{T}}$ 是一个 n 维实数向量, 定义 $\boldsymbol{y} = [y_0, \cdots, y_{n-1}]^{\mathrm{T}} = \boldsymbol{C}\boldsymbol{x}$, 其中 $\boldsymbol{C}$ 是 DCT 矩阵. 设 $1 \leqslant m \leqslant n$ 是另外一个整数, 以 $y_0, \cdots, y_{m-1}$ 为系数的*

$$P_m(t) = \frac{1}{\sqrt{n}}y_0 + \frac{\sqrt{2}}{\sqrt{n}}\sum_{k=1}^{m-1} y_k \cos\frac{k(2t+1)\pi}{2n}$$

极小化了 n 个数据点的平方逼近误差 $\sum\limits_{j=0}^{n-1}(P_m(j) - x_j)^2$.

证 由定理 10.11 直接可得.

在例 11.1 中, 如果要求只用 3 个基函数

$$1, \cos\frac{(2t+1)\pi}{8}, \cos\frac{2(2t+1)\pi}{8}$$

对同样的 4 个数据做最佳二乘逼近, 其结果是

$$P_3(t) = \frac{1}{2}\cdot 0 + \frac{1}{\sqrt{2}}\left[0.923\,9\cos\frac{(2t+1)\pi}{8} + \cos\frac{2(2t+1)\pi}{8}\right]. \tag{11.9}$$

图 11-1 把插值函数 P_4 与最小二乘解 P_3 作了比较.

亮点　正交性

最小二乘逼近的思想是, 找到点到平面 (或子空间) 的最短距离, 即要构造点到平面的垂线. 像第 4 章那样, 这个构造由正规方程组得出. 第 10 章和第 11 章使用这个概念及相对小的一组基函数来尽可能近地逼近数据. 基本的信息是像 DCT 矩阵的行所表现的一样, 要寻找正交的基函数. 于是正规方程组在计算上就变得非常简单 (见定理 10.11).

例 11.2 使用 DCT, 对于 $m = 4, 6, 8$, 求关于数据 $t = 0, \cdots, 7$ 和 $\boldsymbol{x} = [-2.2, -2.8, -6.1, -3.9, 0.0, 1.1, -0.6, -1.1]$ 的最小二乘拟合.

当 $n = 8$ 时, 我们找到数据的 DCT 为

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} = \begin{pmatrix} -5.515\,4 \\ -3.834\,5 \\ 0.583\,3 \\ 4.371\,5 \\ 0.424\,3 \\ -1.55\,4 \\ -0.624\,3 \\ -0.576\,9 \end{pmatrix}.$$

根据定理 11.2, 8 个数据点的 DCT 插值结果为

$$\begin{aligned} P_8(t) = &\frac{1}{\sqrt{8}}(-5.515\,4) + \frac{1}{2}\left[-3.834\,5\cos\frac{(2t+1)\pi}{16} + 0.583\,3\cos\frac{2(2t+1)\pi}{16}\right. \\ &+ 4.371\,5\cos\frac{3(2t+1)\pi}{16} + 0.424\,3\cos\frac{4(2t+1)\pi}{16} \\ &- 1.550\,4\cos\frac{5(2t+1)\pi}{16} - 0.624\,3\cos\frac{6(2t+1)\pi}{16} \end{aligned}$$

$$- 0.576\,9 \cos \frac{7\,(2t+1)\,\pi}{16}.$$

插值结果 P_8 和最小二乘拟合 P_6 和 P_4 画在了图 11-2 上, 后面两个是根据定理 11.3 通过分别只保留了 P_8 前 6 项和前 4 项得到的. ◀

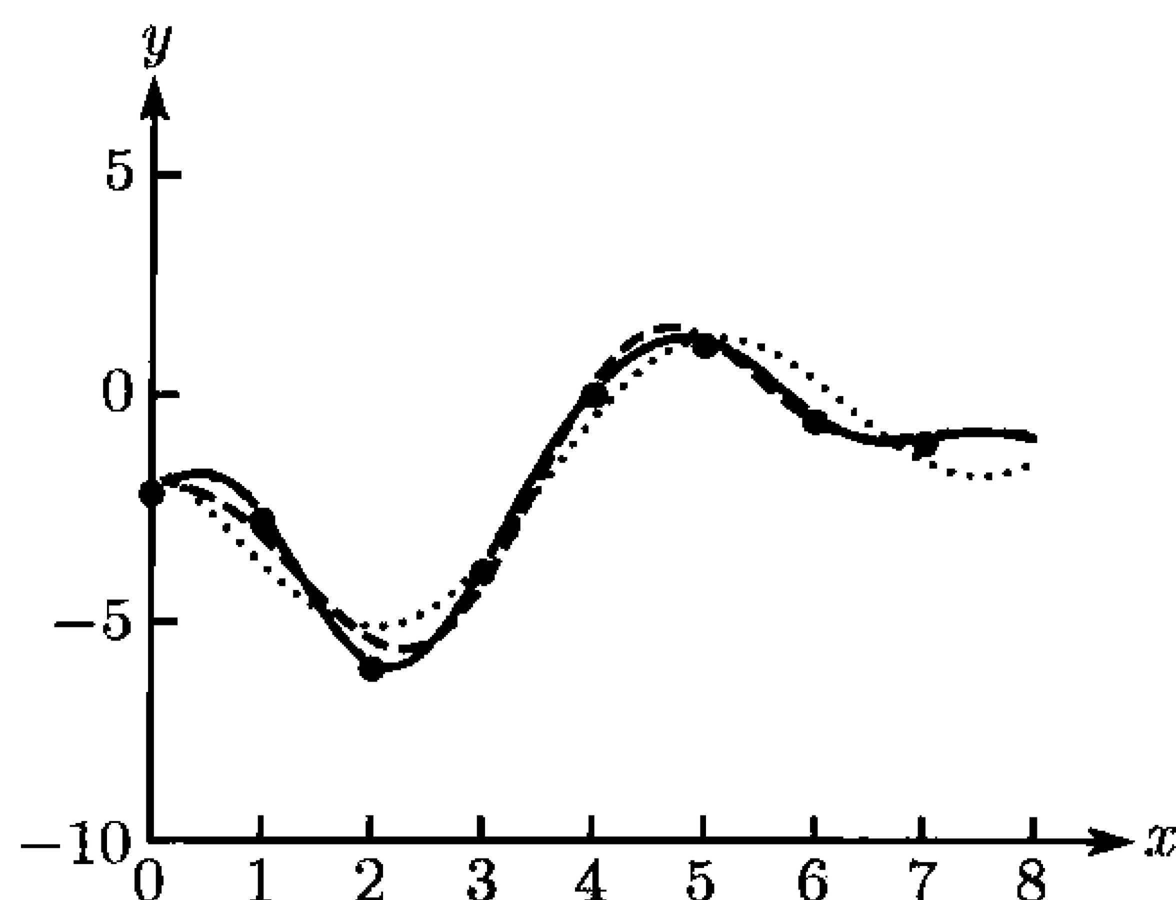


图 11-2 DCT 插值和最小二乘逼近. 实曲线是例 11.2 中数据点的 DCT 插值结果, 分段的虚线是只有开始 6 项的最小二乘拟合, 而点虚线代表 4 项

习题 11.1

1. 用 2×2 的 DCT 矩阵和定理 11.2 来求解数据点的 DCT 插值函数:

(a)

t	x
0	3
1	3

(b)

t	x
0	2
1	−2

(c)

t	x
0	3
1	1

(d)

t	x
0	4
1	−1

2. 根据输入数据 $(0, x_0), (1, x_1)$, 描述 $m = 1$ 的最小二乘 DCT 逼近.

3. 求解下面数据向量 $\boldsymbol{x}$ 的 DCT, 并对数据点 $(i, x_i), i = 1, \cdots, n-1$, 求相应的插值函数 $P_n(t)$. [可以像 (11.7) 中那样定义 b 和 c 来写出答案]

(a)

t	x
0	1
1	0
2	1
3	0

(b)

t	x
0	1
1	1
2	1
3	1

(c)

t	x
0	1
1	0
2	0
3	0

(d)

t	x
0	1
1	2
2	3
3	4

4. 对于习题 3 中的数据, 求解 $m = 2$ 项的 DCT 最小二乘逼近.

5. 写出建立 (11.6) 和 (11.7) 所需要的三角函数运算.

6. (a) 证明对于任何的 x 和 y, 三角函数公式 $\cos(x+y)+\cos(x-y)=2\cos x\cos y$ 成立;

(b) 证明 $\boldsymbol{C}^{\mathrm{T}}$ 的列是 (11.5) 中矩阵 $\boldsymbol{T}$ 的特征向量, 并找出特征值;

(c) 证明 $\boldsymbol{C}^{\mathrm{T}}$ 的列是单位向量.

7. 设计一个和 10.2 节中用来画 DFT 图像类似的简单方法, 来画 DCT 最小二乘多项式 $P_m(t)$ 的图像, 它涉及 DCT 矩阵乘上系数 $\sqrt{p/m}$ 和 DCT 的逆变换.

计算机问题 11.1

1. 画出习题 3 的数据、DCT 插值结果和 $m = 2$ 项的 DCT 最小二乘逼近的图形.

2. 画出下面数据和 $m=4, 6, 8$ 的 DCT 最小二乘逼近的函数图像:

(a)

t	x
0	3
1	5
2	−1
3	3
4	1
5	3
6	−2
7	4

(b)

t	x
0	4
1	1
2	−3
3	0
4	0
5	2
6	−4
7	0

(c)

t	x
0	3
1	−1
2	−1
3	3
4	3
5	−1
6	−1
7	3

(d)

t	x
0	4
1	2
2	−4
3	2
4	4
5	2
6	−4
7	2

3. 在数据点 $(j, f(j)), j=0,\cdots,7$, 画出下列函数 $f(t)$ 和 DCT 插值函数的图像. (a) $f(t)=e^{-t/4}$; (b) $f(t)=\cos\frac{\pi}{2}t$.

11.2 二维 DCT 和图像压缩

二维的 DCT 经常被用来压缩小块图像, 如 8×8 像素的. 这种压缩是有损的, 意思是图像的一些信息会被忽略. DCT 的关键特征是它能帮助组织信息, 使得被忽略的信息恰好是人类的眼睛对它不敏感的. 更确切地说, DCT 向我们展示了如何利用一组基函数, 它们依人类视觉系统所判断的重要性由高到低排列, 对数据进行插值. 那些不太重要的插值项可以根据需要而舍去, 就像报纸编辑在出版期限临近时把长篇故事去掉那样.

接着我们要把所学的关于 DCT 的知识应用到图像压缩中去. 加上量化矩阵的工具和 Huffman 编码, 每个 8×8 的图像可以跟其他图像块一样转化为位流来储存. 当图像需要解压和显示时, 完整的位流通过编码的逆过程被解码. 我们将介绍这种称为标准 JPEG 的方法, 它是用来储存 JPEG 图像的默认方法.

11.2.1 二维 DCT

二维 DCT 只不过是把一维的 DCT 一个接一个应用到二维中. 它可以对给定二维网格上的数据进行插值或逼近, 跟一维的情形很相似. 在图像处理中, 二维网格上的数值表示像素值, 譬如说灰度值、彩色度.

仅在这一章, 当涉及如图 11-3 中所示的二维数据点时, 我们将首先列出纵坐标, 然后列出横坐标. 目标是使得它和矩阵运算的一些通常约定相容, 其中元素 x_{ij} 的下标 i 是沿着纵坐标变化的, j 是沿着横坐标变化的. 这一节的主要应用是可以很自然地把图像视为数的矩阵, 像素文件就表示图像.

图 11-3 显示了在二维平面上每个小矩形网格点 (s_i, t_j) 赋值为 x_{ij} 的 (s,t) 网格点. 为明确起见, 我们将使用整数点网格 $s_i=\{0,1,\cdots,n-1\}$(记住这对应着纵坐标) 和 $t_j=\{0,1,\cdots,n-1\}$(这对应着横坐标). 二维 DCT 的目标是要构造拟合

n^2 个点 (s_i, t_j, x_{ij}) 的插值函数 $F(s,t)$, 其中 $i,j=0,1,\cdots,n-1$. 二维 DCT 在最小二乘意义下以最优方式完成了这一目标, 即说这个拟合很好地舍去了插值函数中部分基函数项.

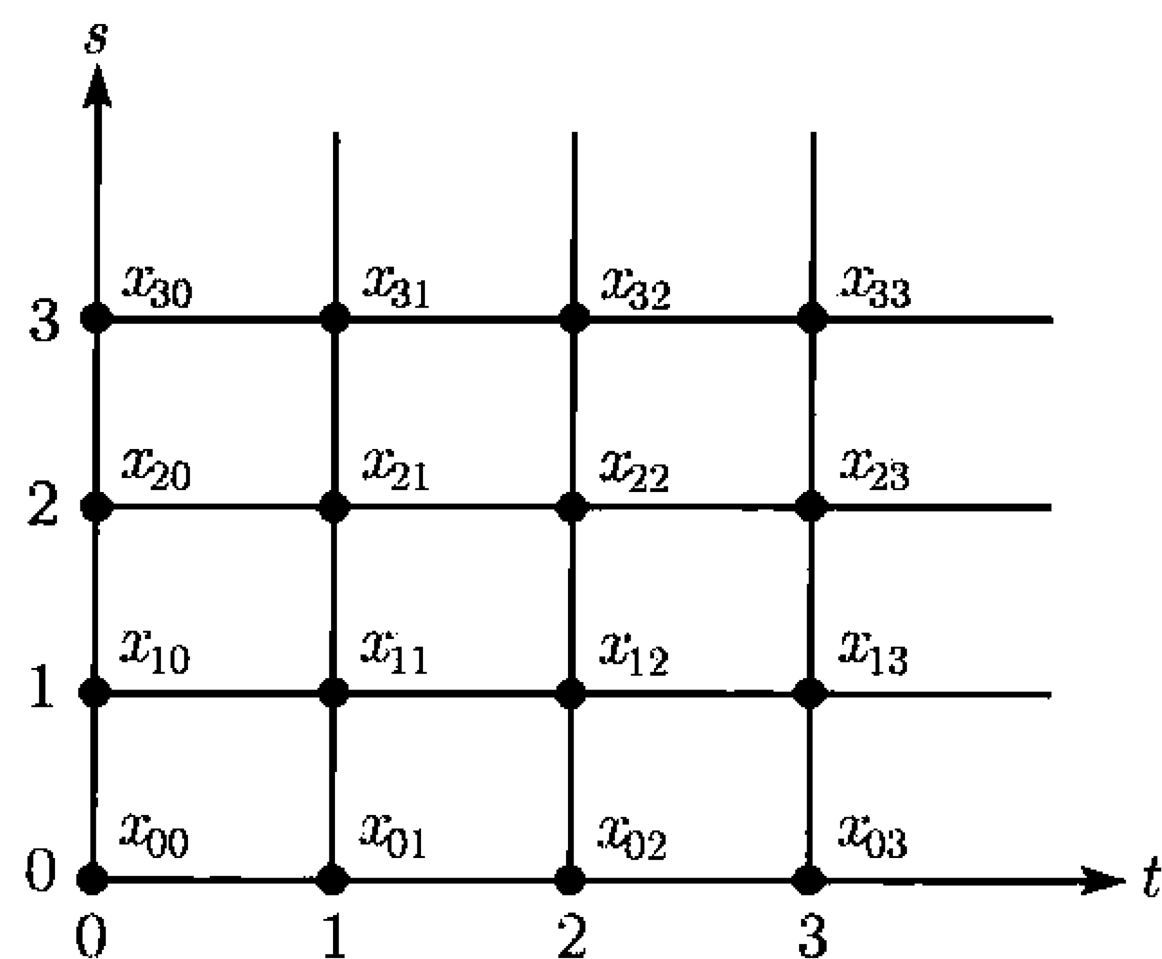


图 11-3　二维网格数据点: 二维的 DCT 能够对正方形网格上的函数值 (如图像的像素值) 进行插值

二维 DCT 是把一维 DCT 连续地应用于纵坐标和横坐标, 考虑以 x_{ij} 为元素值的矩阵 $\boldsymbol{X}$, 如图 11-3 所示. 为把一维 DCT 应用到 s 方向, 我们需要把 $\boldsymbol{X}$ 进行转置, 然后乘上 $\boldsymbol{C}$. 所得到的列恰好是矩阵 $\boldsymbol{X}$ 的行的一维 DCT. $\boldsymbol{C}\boldsymbol{X}^{\mathrm{T}}$ 的某个列刚好对应着某个固定的 t_i. 在 t 方向上做一维 DCT 意味着沿着行进行移动, 因此, 再一次对其进行转置再乘上 $\boldsymbol{C}$ 有

$$\boldsymbol{C}(\boldsymbol{C}\boldsymbol{X}^{\mathrm{T}})^{\mathrm{T}} = \boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}. \tag{11.10}$$

定义 11.4　$n \times n$ 的矩阵 $\boldsymbol{X}$ 的二维 DCT 是 $\boldsymbol{Y} = \boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}$, 其中 $\boldsymbol{C}$ 如 (11.1) 所定义.

例 11.3　解如图 11-4a 中数据的二维 DCT.

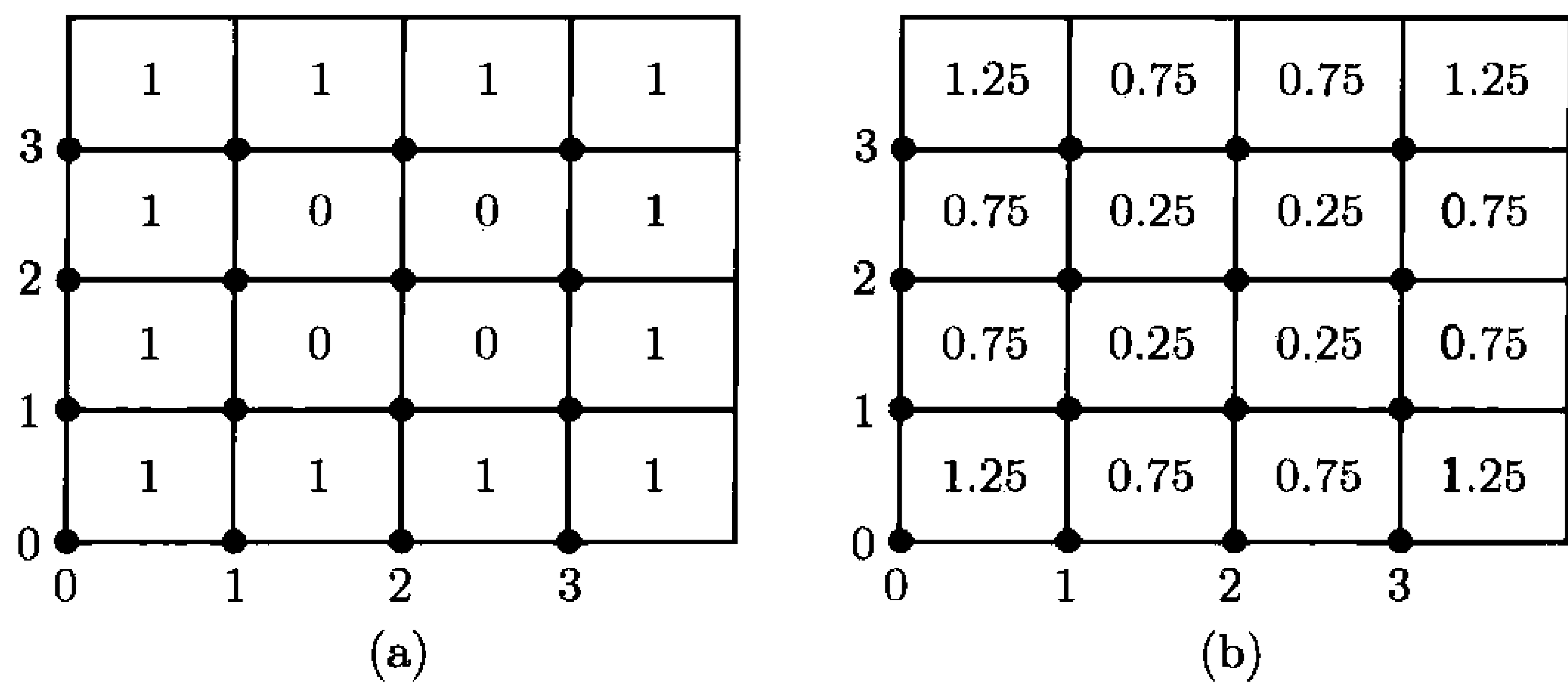


图 11-4　例 11.3 的二维数据 (a)16 个数据点 (i, j, x_{ij}); (b) 式 (11.4) 的最小二乘逼近在网格点上的值

由定义和 (11.6) 式知, 矩阵的二维 DCT 为

$$
\boldsymbol{Y}=\boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}=a\begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} a\begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}
$$

$$
=\begin{bmatrix} 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{11.11}
$$

◀

二维 DCT 的逆变换很容易用 DCT 矩阵 $\boldsymbol{C}$ 来表达. 由于 $\boldsymbol{Y}=\boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}$ 且 $\boldsymbol{C}$ 是正交阵, $\boldsymbol{X}=\boldsymbol{C}^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{C}$ 刚好恢复了图像.

定义 11.5 *$n\times n$ 矩阵 Y 的二维 DCT 的逆是 $\boldsymbol{X}=\boldsymbol{C}^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{C}$.*

像前面所看到的, 正交变换的逆和插值有着紧密的联系. 插值的目标是从由变换产生的插值系数所构造的函数中恢复原始数据点. 由于 $\boldsymbol{C}$ 是一个正交阵, $\boldsymbol{C}^{-1}=\boldsymbol{C}^{\mathrm{T}}$. 由于在等式 $\boldsymbol{X}=\boldsymbol{C}^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{C}$ 中 x_{ij} 可表成余弦函数的乘积, 二维 DCT 的逆也可以写成插值形式.

注意到 (11.1) 中 $\boldsymbol{C}$ 的定义

$$
C_{ij}=\sqrt{\frac{2}{n}}a_i\cos\frac{i(2j+1)\pi}{2n},\quad i,j=0,\cdots,n-1, \tag{11.12}
$$

其中 $a_i\equiv\begin{cases}1/\sqrt{2},\ i=0,\\ 1,i=1,\cdots,n-1\end{cases}$

可以把插值函数写成一种有用的形式. 根据矩阵乘积的法则, 等式 $\boldsymbol{X}=\boldsymbol{C}^{\mathrm{T}}\boldsymbol{Y}\boldsymbol{C}$ 可表达成

$$
\begin{aligned}
x_{ij}&=\sum_{k=0}^{n-1}\sum_{l=0}^{n-1}C^{\mathrm{T}}_{ik}y_{kl}C_{lj}=\sum_{k=0}^{n-1}\sum_{l=0}^{n-1}C_{ki}y_{kl}C_{lj}\\
&=\frac{2}{n}\sum_{k=0}^{n-1}\sum_{l=0}^{n-1}y_{kl}a_k a_l\cos\frac{k(2i+1)\pi}{2n}\cos\frac{l(2j+1)\pi}{2n}.
\end{aligned} \tag{11.13}
$$

这正是我们要找的插值函数形式.

定理 11.6 二维 DCT 插值定理 *设 $\boldsymbol{X}=(x_{ij})$ 是 $n\times n$ 个实数的矩阵, $\boldsymbol{Y}=(y_{kl})$ 是 $\boldsymbol{X}$ 的二维 DCT, 定义 $a_0=1/\sqrt{2}$ 和 $a_k=1,k>0$, 则实函数*

$$
P_n(s,t)=\frac{2}{n}\sum_{k=0}^{n-1}\sum_{l=0}^{n-1}y_{kl}a_k a_l\cos\frac{k(2s+1)\pi}{2n}\cos\frac{l(2t+1)\pi}{2n}
$$

满足 $P_n(i,j)=x_{ij}$, 其中 $i,j=0,\cdots,n-1$.

回到例 11.3, 不为 0 的插值系数仅有 $y_{00}=3, y_{02}=y_{20}=1$, $y_{22}=-1$. 由定理 11.6 写出的插值函数为

$$\begin{aligned}P_4(s,t)=&\frac{2}{4}\left[\frac{1}{2}y_{00}+\frac{1}{\sqrt{2}}y_{02}\cos\frac{2\,(2t+1)\,\pi}{8}+\frac{1}{\sqrt{2}}y_{20}\cos\frac{2\,(2s+1)\,\pi}{8}\right.\\&\left.+y_{22}\cos\frac{2\,(2s+1)\,\pi}{8}\cos\frac{2(2t+1)\pi}{8}\right]\\=&\frac{1}{2}\left[\frac{1}{2}\,(3)+\frac{1}{\sqrt{2}}\,(1)\cos\frac{2\,(2t+1)\,\pi}{8}+\frac{1}{\sqrt{2}}\,(1)\cos\frac{2\,(2s+1)\,\pi}{8}\right.\\&\left.+(-1)\cos\frac{2\,(2s+1)\,\pi}{8}\cos\frac{2\,(2t+1)\,\pi}{8}\right]\\=&\frac{3}{4}+\frac{1}{2\sqrt{2}}\cos\frac{2\,(2t+1)\,\pi}{4}+\frac{1}{2\sqrt{2}}\cos\frac{2\,(2s+1)\,\pi}{4}\\&-\frac{1}{2}\cos\frac{(2s+1)\,\pi}{4}\cos\frac{(2t+1)\,\pi}{4}.\end{aligned}$$

举例验证一下插值, 如 $P_4(0,0)=\frac{3}{4}+\frac{1}{4}+\frac{1}{4}-\frac{1}{4}=1$, $P_4(1,2)=\frac{3}{4}-\frac{1}{4}-\frac{1}{4}-\frac{1}{4}=0$, 和图 11-4 中的数据一样. 插值函数中的常数项 y_{00}/n 被称为展式中的 “DC” 部分 (direct current), 它是数据的简单平均, 非常数项包含了数据关于平均值的扰动. 在这个例子中, 12 个 1 和 4 个 0 的平均是 $y_{00}/4=3/4$.

二维 DCT 的最小二乘逼近和一维 DCT 中的做法一样. 例如, 使用一个低通过滤器将会除去高频部分, 即插值函数中其系数具有较大下标的项. 在例 11.3 中, 对于 $i+j\leqslant 2$ 的基函数

$$\cos\frac{i(2s+1)\pi}{8}\cos\frac{j(2t+1)\pi}{8}$$

的最佳最小二乘拟合是去掉那些不满足 $i+j\leqslant 2$ 的项. 在这种情况下, 仅有 $i=j=2$ 是非零的高频项, 得到

$$P_2(s,t)=\frac{3}{4}+\frac{1}{2\sqrt{2}}\cos\frac{(2t+1)\pi}{4}+\frac{1}{2\sqrt{2}}\cos\frac{(2s+1)\pi}{4}. \tag{11.14}$$

这种最小二乘逼近如图 11-4b 所示.

对于向量 $\boldsymbol{x}$ 的一维 DCT 的 MATLAB命令是

```
>> y=dct(x);
```

dct的输入可以是一个向量或一个矩阵. 在后一种情况下, 返回的结果是矩阵每一列的 DCT. 令人惊讶的是, MATLAB不是通过乘上适当的 DCT 矩阵 C, 而是

通过快速 Fourier 变换, 来计算 DCT 的. 当然 MATLAB也能通过把变换作用于单位阵来显示 $n \times n$ 的 DCT 矩阵:

```
>> C=dct(eye(n))
```

为使得 MATLAB执行二维 DCT, 我们回到 (11.10) 中的定义. 对于矩阵 $\boldsymbol{X}$ 命令

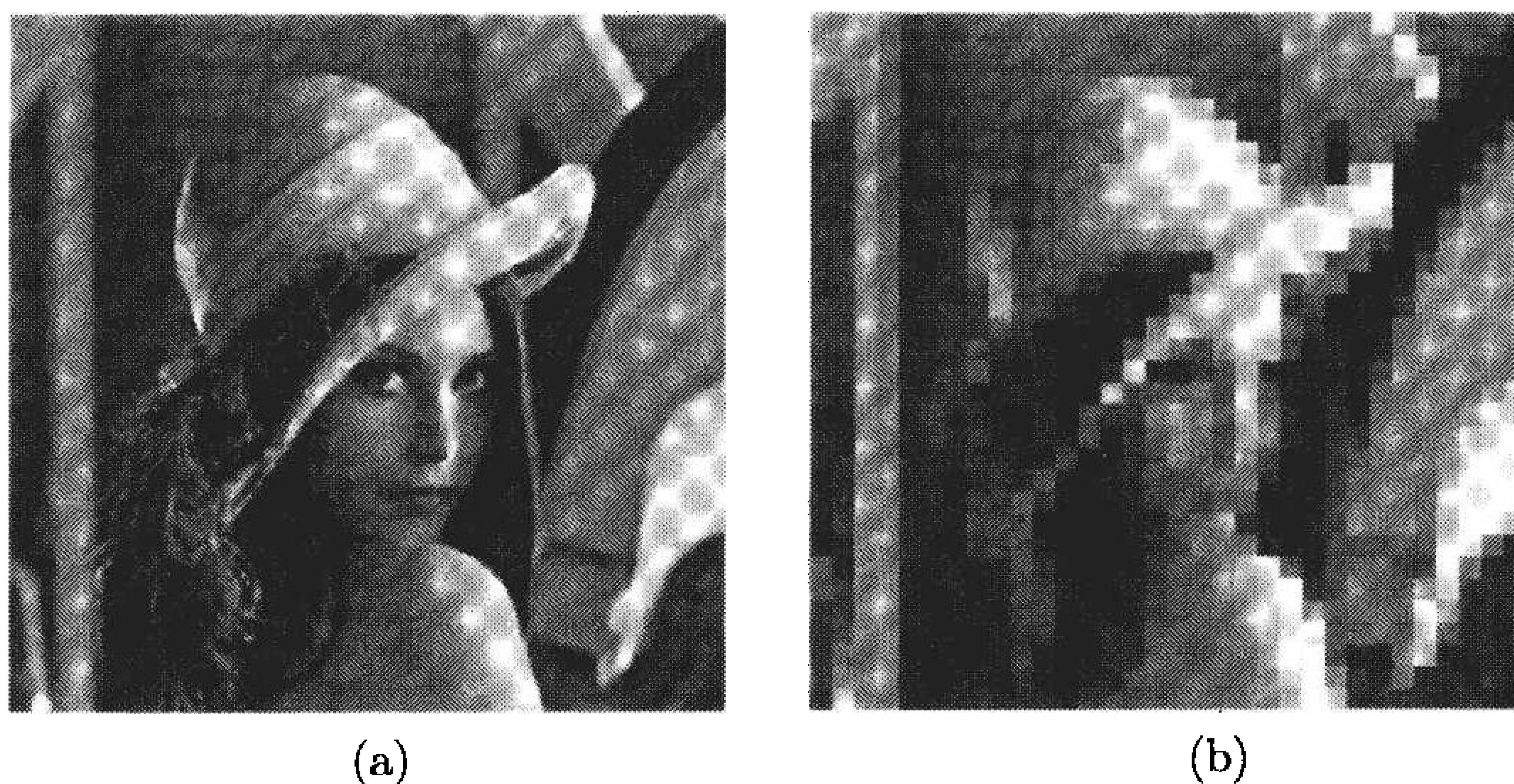
```
>> Y=dct(dct(X')')
```

两次使用一维 DCT 计算二维 DCT.

11.2.2 图像压缩

DCT 中所讲到的正交性对于实施图像压缩至关重要. 图像中包含由数 (对于彩色图像而言是向量) 来表示的像素. 简便的方法如 DCT 能很容易地做最小二乘逼近使得需要表示像素值的位数减少, 但原图像只是很轻微的失真, 并且肉眼可能觉察不到.

图 11-5a 显示了一幅 256×256 像素的灰度图像. 每个像素的灰度值由一个字节来表示, 一个字节有 8 位, 表示整数从 0 到 255. 当为 0 时是黑色, 255 时是白色. 我们可以认为图中所示的信息为 256×256 的整数数组. 以这种方式来表示, 该图像保存有 $256 \times 256 = 2^{16} = 64$ K 字节的信息.



(a) (b)

图 11-5 灰度图像.(a)256×256 网格的每一个像素由 0~255 中的一个整数表示; (b) 粗糙的压缩, 每个 8×8 的像素块由它们的灰度平均值来表示

MATLAB通过使用标准的图像格式来导入图像的灰度值或 RGB 值. 例如给定一个图像文件picture.jpg, 命令

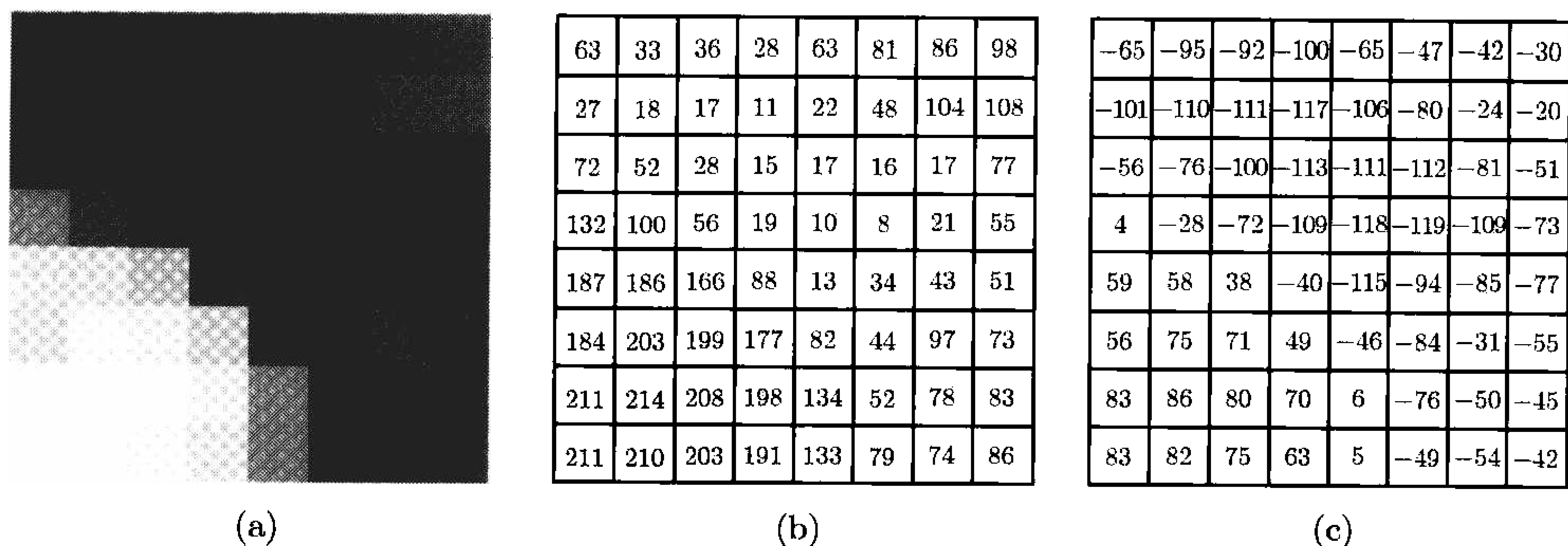
```
» xin = imread('picture.jpg');
» x = double(xin);
```

把灰度值矩阵输入到双精度变量 x 中. 如果图像是 RGB 彩色的, 数组变量就会是用 3 维的向量来表示 3 种颜色. 本节主要讨论灰度值将其拓展到彩色图像是

很直接的, 我们将在本节最后对它进行讨论. MATLAB命令imagesc(x) 将把数组呈现为灰度图像或彩色图像, 依赖于 x 是二维的或三维的.

图 11-5b 显示了用一种粗糙的压缩方法压缩图像的结果, 其中每个 8×8 像素块的像素值由它们的平均值来代替. 数据的压缩量是很可观的, 因为只剩下 $32^2 = 2^{10}$ 个像素块, 其中每块的像素值由一个整数表示——但是压缩所得的图像质量很糟糕. 我们的目标是通过使用几个整数代替每一个 8×8 的像素块来更好地表达原始图像的信息而不把图像压缩得那么厉害.

开始我们把问题简化为 8×8 的像素块, 如图 11-6a 所示. 这个块取自于图 11-5 的右眼区域. 图 11-6b 用各占一个字节的整数来表示了 64 个像素的灰度强度. 图 11-6c 中, 我们把每个像素值减去 256/2=128, 使得这些数据以 0 点为中心. 这一步并不是必需的, 只是这样中心化后能更好地使用二维 DCT.



(b)

63	33	36	28	63	81	86	98
27	18	17	11	22	48	104	108
72	52	28	15	17	16	17	77
132	100	56	19	10	8	21	55
187	186	166	88	13	34	43	51
184	203	199	177	82	44	97	73
211	214	208	198	134	52	78	83
211	210	203	191	133	79	74	86

(c)

−65	−95	−92	−100	−65	−47	−42	−30
−101	−110	−111	−117	−106	−80	−24	−20
−56	−76	−100	−113	−111	−112	−81	−51
4	−28	−72	−109	−118	−119	−109	−73
59	58	38	−40	−115	−94	−85	−77
56	75	71	49	−46	−84	−31	−55
83	86	80	70	6	−76	−50	−45
83	82	75	63	5	−49	−54	−42

图 11-6 8×8 像素块的例子: (a) 灰度显示; (b) 灰度像素值; (c) 灰度像素值 −128

为了压缩所示的 8×8 的像素块, 我们将对灰度值矩阵 $\boldsymbol{X}$ 进行变换, 其中

$$X = \begin{bmatrix} -65 & -95 & -92 & -100 & -65 & -47 & -42 & -30 \\ -101 & -110 & -111 & -117 & -106 & -80 & -24 & -20 \\ -56 & -76 & -100 & -113 & -111 & -112 & -81 & -51 \\ 4 & -28 & -72 & -109 & -118 & -119 & -107 & -73 \\ 59 & 58 & 38 & -40 & -115 & -94 & -85 & -77 \\ 56 & 75 & 71 & 49 & -46 & -84 & -31 & -55 \\ 83 & 86 & 80 & 70 & 6 & -76 & -50 & -45 \\ 83 & 82 & 75 & 63 & 5 & -49 & -54 & -42 \end{bmatrix}, \tag{11.15}$$

这依赖于二维 DCT 根据信息对人类视觉系统的重要性进行选择信息的能力.

我们计算 $\boldsymbol{X}$ 的二维 DCT, 为简单计经四舍五入得

$$Y = C_8 X C_8^{\mathrm{T}} = \begin{bmatrix} -304 & 210 & 104 & -69 & 10 & 20 & -12 & 7 \\ -327 & -260 & 67 & 70 & -10 & -15 & 21 & 8 \\ 93 & -84 & -66 & 16 & 24 & -2 & -5 & 9 \\ 89 & 33 & -19 & -20 & -26 & 21 & -3 & 0 \\ -9 & 42 & 18 & 27 & -7 & -17 & 29 & -7 \\ -5 & 15 & -10 & 17 & 32 & -15 & -4 & 7 \\ 10 & 3 & -12 & -1 & 2 & 3 & -2 & -3 \\ 12 & 30 & 0 & -3 & -3 & -6 & 12 & -1 \end{bmatrix}. \tag{11.16}$$

这样的四舍五入带来了小量的额外误差且不是严格必需的, 但是它会使压缩变得更容易. 注意到有这样的一种趋势, 与右下方相比, 更多的信息要被储存在变换矩阵 Y 的左上方. 右下方表示了对视觉系统而言不那么重要的高频基函数. 然而, 由于二维 DCT 是一个可逆变换, Y 中的信息完全可以重建原始图像, 达到四舍五入的标准.

低通过滤器是我们选择的第一种压缩策略. 如 12.1 节所讨论的, 二维 DCT 的最小二乘逼近仅是把插值函数 $P_8(s,t)$ 的一些项舍去. 如我们可以通过令 $y_{kl}=0$(当 $k+1 \geqslant 7$ 时), 把具有相对高空间频率的函数所起的影响去掉, 经过低通过滤, 变换的系数为

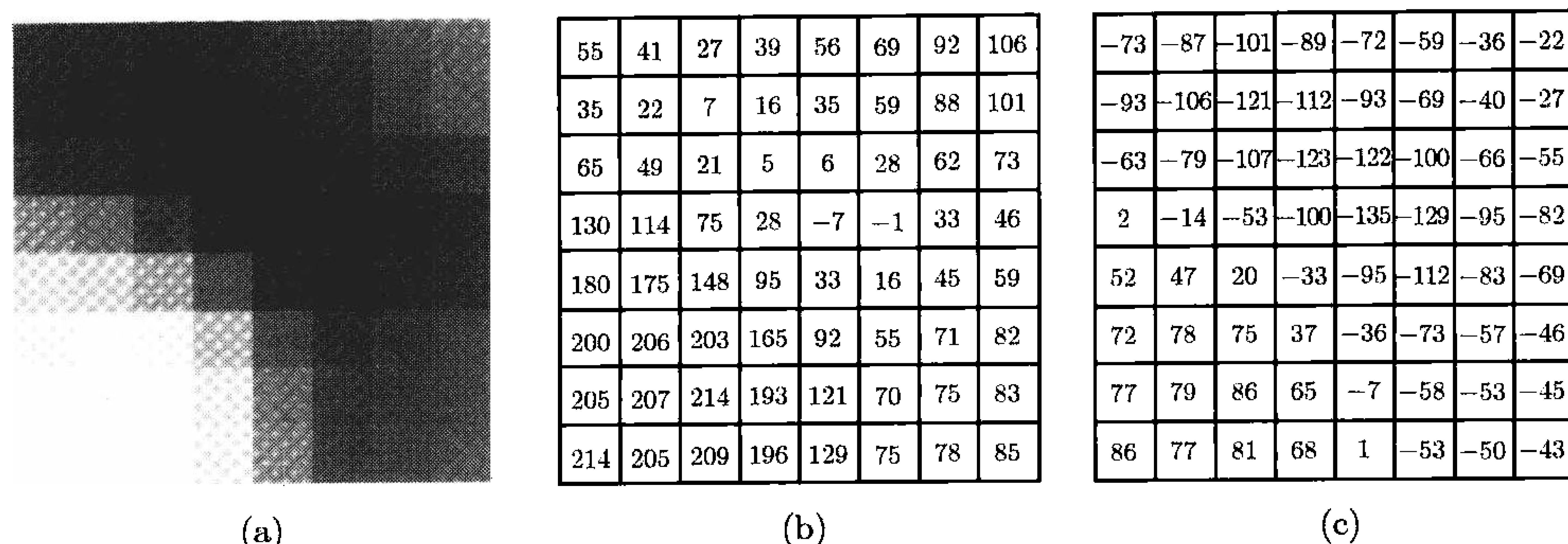
$$Y_{\mathrm{low}} = \begin{bmatrix} -304 & 210 & 104 & -69 & 10 & 20 & -12 & 0 \\ -327 & -260 & 67 & 70 & -10 & -15 & 0 & 0 \\ 93 & -84 & -66 & 16 & 24 & 0 & 0 & 0 \\ 89 & 33 & -19 & -20 & 0 & 0 & 0 & 0 \\ -9 & 42 & 18 & 0 & 0 & 0 & 0 & 0 \\ -5 & 15 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{11.17}$$

为构造图像, 我们使用二维 DCT 的逆, 如 $C_8^{\mathrm{T}} Y_{\mathrm{low}} C_8$, 获得的灰度像素值如图 11-7 所示, 图 11-7a 的图像和图 11-6a 的原始图像很近似, 但细节上是不同的.

那么我们从 8×8 像素块压缩了多少信息呢? 如果不考虑四舍五入的话, 原始图像可以通过对 (11.6) 做二维 DCT 的逆变换再加上 128 来进行重构. 经过低通过滤器后, 减少了差不多一半的存储量, 但是保留了像素块大部分的视觉信息.

11.2.3 量化

量化的想法允许我们以一种更有选择的方式使用低通过滤器来达到它的效果. 我们将在更低的存储成本下保留一些系数的低精度版本, 而不是完全忽略掉它们.



(a)

55	41	27	39	56	69	92	106
35	22	7	16	35	59	88	101
65	49	21	5	6	28	62	73
130	114	75	28	−7	−1	33	46
180	175	148	95	33	16	45	59
200	206	203	165	92	55	71	82
205	207	214	193	121	70	75	83
214	205	209	196	129	75	78	85

(b)

−73	−87	−101	−89	−72	−59	−36	−22
−93	−106	−121	−112	−93	−69	−40	−27
−63	−79	−107	−123	−122	−100	−66	−55
2	−14	−53	−100	−135	−129	−95	−82
52	47	20	−33	−95	−112	−83	−69
72	78	75	37	−36	−73	−57	−46
77	79	86	65	−7	−58	−53	−45
86	77	81	68	1	−53	−50	−43

(c)

图 11-7 使用低通过滤器的结果: (a) 过滤后的图像; (b) 经变换再加上 128 后的灰度值; (c) 逆变换后的数据

这种想法利用了人类视觉系统的相同方面, 即对更高空间频率反而不敏感. 主要思想是分配少一些的位数来存储变换矩阵 $\boldsymbol{Y}$ 右下角的信息而不是把它丢掉.

模 q 的量化

量化: $z = \text{round}(y/q)$ (11.18)

去量化: $\bar{y} = qz$

这里的 "round" 是 "四舍五入" 的意思, **量化误差**(quantization error) 是输入的 y 与经量化和去量化处理后而输出的 $\bar{y}$ 的差. 模 q 数量化的最大误差是 $q/2$.

例 11.4 对 $-10, 3, 65$ 进行模 8 的量化.

量化后的值为 $-1, 0, 8$. 去量化后结果是 $-8, 0, 64$, 各自的误差为 $|-2|, |3|, |1|$ 每个均小于 $q/2 = 4$. ◀

回到图像的例子, 每个频率所允许的位数可以任意选取. 称 8×8 的矩阵 $\boldsymbol{Q}$ 为**量化矩阵**. 元素 q_{kl}, 其中 $0 \leqslant k, l \leqslant 7$, 将限制我们分配给变换所得矩阵 $\boldsymbol{Y}$ 每个位置上元素的位数. 用压缩矩阵

$$\boldsymbol{Y_Q} = \text{round}[ykl/qkl] \tag{11.19}$$

来代替 $\boldsymbol{Y}$. 矩阵 $\boldsymbol{Y}$ 的每个位置上元素除以量化矩阵相应位置上的元素. 信息的丢失发生在接下来所做的四舍五入, 它使得这种方法成为有损压缩, 可以看到, 量化矩阵中元素值越大信息丢失得越多.

作为第一个例子, **线性量化**(linear quantization) 由如下矩阵定义:

$$q_{kl} = 8p(k + l + 1), \qquad 0 \leqslant k, l \leqslant 7, \tag{11.20}$$

其中 p 是常数, 称为**损失参数**(loss parameter). 于是

$$
Q = p\begin{bmatrix} 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 \\ 16 & 24 & 32 & 40 & 48 & 56 & 64 & 72 \\ 24 & 32 & 40 & 48 & 56 & 64 & 72 & 80 \\ 32 & 40 & 48 & 56 & 64 & 72 & 80 & 88 \\ 40 & 48 & 56 & 64 & 72 & 80 & 88 & 96 \\ 48 & 56 & 64 & 72 & 80 & 88 & 96 & 104 \\ 56 & 64 & 72 & 80 & 88 & 96 & 104 & 112 \\ 64 & 72 & 80 & 88 & 96 & 104 & 112 & 120 \end{bmatrix}
$$

损失参数越小, 重构的结果越好. 矩阵 $\boldsymbol{Y}_Q$ 里数的结果集合表示一张新的图像.

为解压缩文件, 通过逆过程对 $\boldsymbol{Y}_Q$ 进行去量化, 将其上每个位置上的元素乘上 $\boldsymbol{Q}$ 相应位置上的元素. 这是图像编码的损失部分. 将 y_{kl} 除以 q_{kl} 再进行四舍五入, 然后通过乘以 q_{kl} 进行重构, 这可能给 y_{kl} 加上 $q_{kl}/2$ 的误差, 这就是量化误差. q_{kl} 越大, 在重构图像时潜在的误差也就越大; 另一方面, q_{kl} 越大, $\boldsymbol{Y}_Q$ 里的整数元素会越小, 那么只需更少的位数来储存它们. 这是在图像的精确度和文件大小中进行的权衡.

事实上, 量化完成了两件事: 高频中只有很小影响的很多元素通过 (11.19) 直接设为 0, 那些不为 0 的也变得更小, 这使得只要很少的位数来存储和传输它们. 这些数将通过 Huffman 编码 (11.3 节介绍) 来转换为位流.

损失参数 p 是用来权衡存储量和视觉精确度的旋钮. 为使用线性量化矩阵, 将元素 y_{kl} 除以 $8p(k+l+1)$, 然后四舍五入取整数. 在 MATLAB中这相当容易:

```
>> Yq = round(Y.*hilb(8)/(8*p));
```

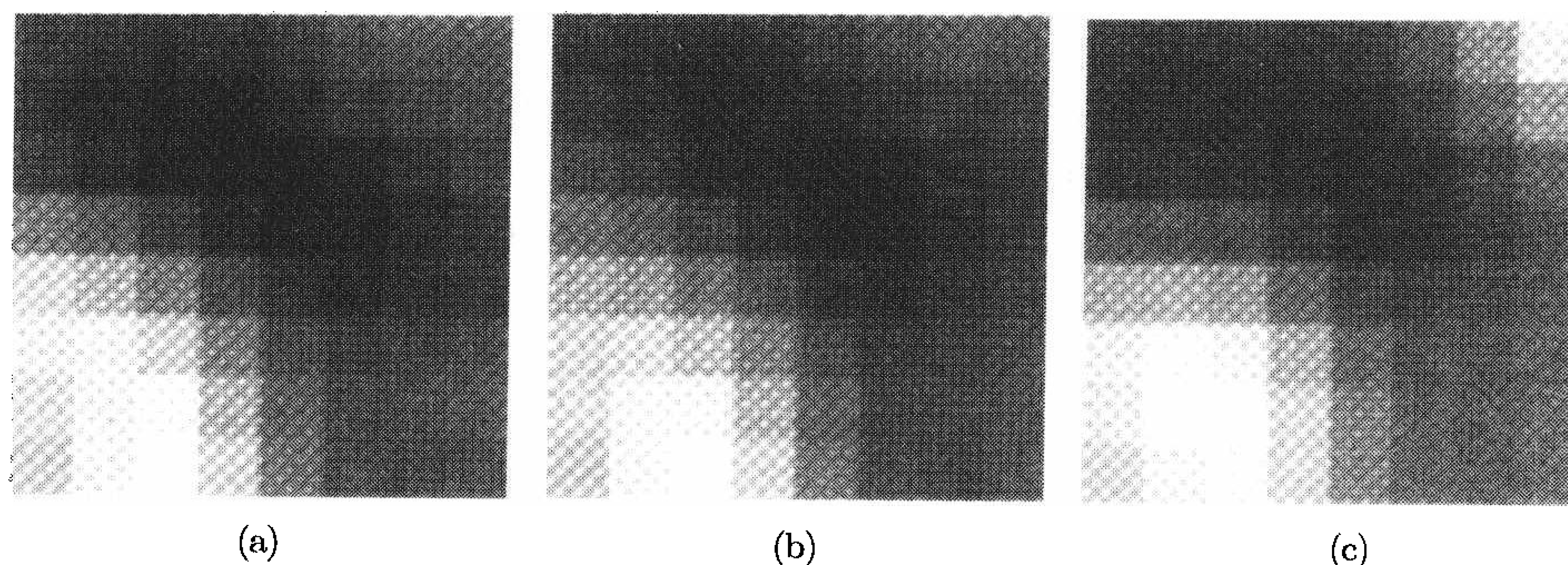
$p=1$ 的线性量化矩阵用到 (11.16) 中, 所得的系数矩阵为

$$
\begin{bmatrix} -38 & 13 & 4 & -2 & 0 & 0 & 0 & 0 \\ -20 & -11 & 2 & 2 & 0 & 0 & 0 & 0 \\ 4 & -3 & -2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{11.21}
$$

为重构图像必须将变换所得的矩阵乘上 $\boldsymbol{Q}$, 并且重新变回像素值. 在 MATLAB中, 使用前面命令的逆过程有

```
>> Xq = 8*p* Yq./hilb(8);
```

使用 $p=1$ 的 $\boldsymbol{Y}_Q$ 重构的图像块如图 11-8a 所示. 与原始的图像块相比只有略微的不同, 并且比使用低通过滤器重构的结果更接近于原始图像.



(a) (b) (c)

图 11-8 线性量子化的结果: 损失参数为 (a)$p=1$, (b)$p=2$, (c)$p=4$

用 $p=2$ 的线性量化矩阵量化后的系数矩阵为

$$Y_Q=\begin{bmatrix} -19 & 7 & 2 & -1 & 0 & 0 & 0 & 0 \\ -10 & -5 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{11.22}$$

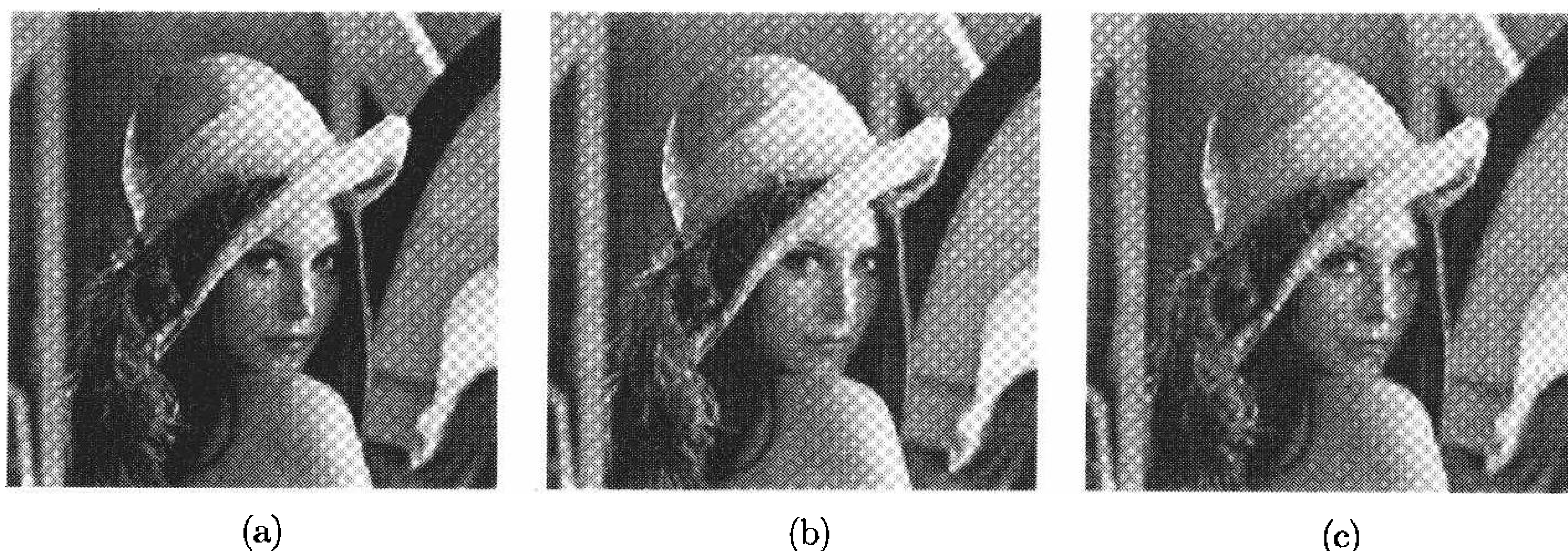
用 $p=4$ 的线性量化矩阵量化后的系数矩阵为

$$Y_Q=\begin{bmatrix} -9 & 3 & 1 & -1 & 0 & 0 & 0 & 0 \\ -5 & -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{11.23}$$

图 11-8 显示了使用 3 个不同损失参数 p 的量化结果. 注意到损失参数 p 的值越大, 通过量化过程矩阵 Y_Q 中被置为 0 的元素越多, 表示像素所需的数据越小, 那么重构的图像越不忠实于原来图像.

接下来我们对图 11-5 中的 1 024 块图像进行量化. 损失参数为 $p=1, 2, 4$ 的结果如图 11-9 所示. 当 $p=4$ 时图像已经被严重损坏.

我们可以粗略地计算一下量化方法造成的图像的压缩量. 原始图像使用从 0 到 255 中的整数来表示像素值, 每个整数占用一个字节或 8 位. 对于 8×8 的像素块, 未经压缩所需的位总数总计是 $8(8)^2 \approx 512$ 位.



(a) (b) (c)

图 11-9 对所有 1 024 个 8×8 图像块做线性量化的结果: 损失参数为 (a)$p=1$, (b)$p=2$, (c)$p=4$

现假设使用损失参数 $p=1$ 的线性量化矩阵. 假设二维 DCT 变换得的矩阵 $\boldsymbol{Y}$ 的最大元素为 255, 然后经 $\boldsymbol{Q}$ 量子化, $\boldsymbol{Y_Q}$ 的最大可能元素为

$$\begin{bmatrix} 32 & 16 & 11 & 8 & 6 & 5 & 5 & 4 \\ 16 & 11 & 8 & 6 & 5 & 5 & 4 & 4 \\ 11 & 8 & 6 & 5 & 5 & 4 & 4 & 3 \\ 8 & 6 & 5 & 5 & 4 & 4 & 3 & 3 \\ 6 & 5 & 5 & 4 & 4 & 3 & 3 & 3 \\ 5 & 5 & 4 & 4 & 3 & 3 & 3 & 2 \\ 5 & 4 & 4 & 3 & 3 & 3 & 2 & 2 \\ 4 & 4 & 3 & 3 & 3 & 2 & 2 & 2 \end{bmatrix}.$$

因为正的和负的元素均有可能出现, 所需用来存储各元素的位数是

$$\begin{bmatrix} 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 \\ 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 4 & 4 & 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 4 & 4 & 4 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

这 64 个数的总和为 249, 或 149/64≈3.89 位/像素, 它比存储 8×8 图像矩阵的原像素值所需的位数的一半还少. 对于 p 的其他值的相应统计结果, 如下表所示:

p	总位数	位/像素
1	249	3.89
2	191	2.98
4	147	2.30

从上表我们可以看到, 当 $p=1$ 时, 只有一些很细微的变化可以看出来, 但是表示图像所需的位数却减少了 1/2. 这是由量化得来的压缩. 为进一步压缩, 我们要利用许多高频项在变换中经量化后变为 0 的这个事实. 通过 11.3 节所介绍的 Huffman 编码和游程编码 (run-length coding), 这将能够非常有效地做到.

$p=1$ 的线性量化非常接近于默认的 JPEG 量化. 量化矩阵提供了有最小图像失真的大部分压缩, 它已经成为很多研究讨论的主题. JPEG 标准包含了一个称为 "Annex K: Examples and Guidelines" 的附录, 它包含了一个基于人类视觉系统试验的 $\boldsymbol{Q}$. 矩阵

$$\boldsymbol{Q_Y}=p\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{11.24}$$

广泛地使用于当前发行的 JPEG 编码器中. 设损失参数 $p=1$, 它就人类视觉系统而言提供了事实上完美的重构, 而当 $p=4$ 时, 却经常发生显著的失真. 在某种程度上, 视觉质量依赖于像素的大小: 当像素很小时, 一些误差可能不被发觉.

到现在为止, 我们仅仅讨论了灰度值图像. 很容易把前面的方法推广到彩色图像 (可以用 RGB 彩色系统表示) 中去. 每一个像素被赋予 3 个整数, 分别表示红、绿、蓝的强度. 一种方法是把压缩过程独立地作用于每一种颜色, 把它们当作灰度图像来处理, 最后从这 3 种颜色重构图像.

尽管 JPEG 标准没有说如何处理颜色, 但是作为基准的 JPEG 采取了一种更为细致的方法. 定义**亮度**(luminance)$Y=0.299R+0.587G+0.114B$, **颜色差**(color difference)$U=B-Y$ 和 $V=R-Y$. 这把 RGB 颜色数据转换为 YUV 系统. 由于 RGB 值可以通过 $B=U+Y, R=V+Y$ 和 $G=(Y-0.299R-0.114B)/0.587$ 重新算出, 所以这是完全可逆的变换. 基准的 JPEG 把前面介绍的 DCT 过滤器独立

地用到 Y, U, V 中, 对于亮度使用附录 Annex K 中的量化矩阵 $\boldsymbol{Q}_Y$, 对于颜色差 U 和 V 使用量化矩阵

$$\boldsymbol{Q}_C = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}. \tag{11.25}$$

经过重构 Y, U 和 V, 它们可以重新转换回 RGB 来重构图像.

由于 U, V 在人类视觉系统中相对地不是那么重要, 所以可以使用更积极的量化方法, 如式 (11.25) 所示. 进一步的压缩可以使用一些特别的技巧得到——例如通过平均色差和在更粗精度的网格上处理它们.

习题 11.2

1. 求解下列数据矩阵 $\boldsymbol{X}$ 的二维 DCT, 并且为数据点 $(i, j, x_{ij}), i, j = 0, 1$ 找到相应的插值函数 $P_2(s, t)$:

(a) $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$; (d) $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

2. 求解下列数据矩阵 $\boldsymbol{X}$ 的二维 DCT, 并且为数据点 $(i, j, x_{ij}), i, j = 0, \cdots, n-1$ 找到相应的插值函数 $P_n(s, t)$:

(a) $\begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$; (c) $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$;

(d) $\begin{bmatrix} 3 & 3 & 3 & 3 \\ 3 & -1 & -1 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & -1 & -1 & 3 \end{bmatrix}$.

3. 用基函数 $1, \cos\frac{(2s+1)\pi}{8}, \cos\frac{(2t+1)\pi}{8}$, 求解习题 2 中的数据的最小二乘逼近.

4. 使用量化矩阵 $\boldsymbol{Q} = \begin{pmatrix} 10 & 20 \\ 20 & 100 \end{pmatrix}$ 对下列矩阵进行量化. 写出量化后的矩阵、(有损的) 去量化后的矩阵和量化误差矩阵.

计算机问题 11.2

1. 求解下列数据矩阵 $\boldsymbol{X}$ 的二维 DCT:

(a) $\begin{bmatrix} -1 & 1 & -1 & 1 \\ -2 & 2 & -2 & 2 \\ -3 & 3 & -3 & 3 \\ -4 & 4 & -4 & 4 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 2 & -1 & -2 \\ -1 & -2 & 1 & 2 \\ 1 & 2 & -1 & -2 \\ -1 & -2 & 1 & 2 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & 3 & 1 & -1 \\ 2 & 1 & 0 & 1 \\ 1 & -1 & 2 & 3 \\ 3 & 2 & 1 & 0 \end{bmatrix}$;

(d) $\begin{bmatrix} -3 & -2 & -1 & 0 \\ -2 & -1 & 0 & 1 \\ -1 & 0 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$.

2. 使用计算机问题 1 中的二维 DCT, 通过设 $k+l \geqslant 4$ 时所有的 $Y_{kl}=0$, 找出 $\boldsymbol{X}$ 的最小二乘低通过滤的逼近.
3. 获得所选择的一个灰度图像文件, 用`imread`命令把它读到 MATLAB中. 裁减所得矩阵使得它的每一维度都恰好是 8 的倍数.(如果需要, 把一个 RGB 彩色图像文件通过标准公式 $X=0.212\,6\,R+0.715\,2\,G+0.072\,2\,B$ 转换为灰度图像.)
   (a) 抽取一个 8×8 的像素块, 通过使用 MATLAB命令`xb=x(81: 88, 81: 88)`. 使用命令`imagesc`显示图像块.
   (b) 应用二维 DCT.
   (c) 使用 $p=1, 2, 4$ 的线性量化矩阵进行量化.
   (d) 通过使用二维 DCT 的逆对图像块进行重构, 并和原始图像进行比较.
   (e) 对所有 8×8 的像素块应用 (a)~(d) 的过程, 并在每一种情况下重建图像.
4. 使用 JPEG 所建议的 $p=1$ 时的矩阵 (11.24) 为量化矩阵, 重新执行计算机问题 3 中的步骤.
5. 获得所选择的一个彩色图像文件. 使用线性量化分别对 R,G,B 实施计算机问题 3 中的过程, 然后重新组合成彩色图像.
6. 获得一个彩色图像, 然后把 RGB 值转换到亮度/色差坐标系统下. 使用 JPEG 量化矩阵分别对 Y, U, V 实施计算机问题 3 中的过程, 并重新组合成彩色图像.

11.3 Huffman 编码

有损的图像压缩需要在图像精确度和文件大小中做权衡. 如果图像精确度损失足够小以至于对于图像的某种应用来说很难察觉, 那么这种让步是值得的. 精确度的损失发生在量化那一步, 在把图像转换分成空间频率后. 无损的压缩是指进一步的不损失一点精度的压缩, 即指对经 DCT 变换和量化后的矩阵进行有效的编码.

本节讨论图像的无损压缩. 作为一个相关的应用, 我们有简单有效的方法把 11.2 节所讲到的量化的 DCT 变换矩阵转换成 JPEG 位流. 学习这些将使我们粗略地知道一些基本的信息论.

11.3.1 信息论和编码

我们考虑一串符号组成的信息. 符号是任意的, 假设它们来自有限集. 在本节里, 我们用一些有效的方式来编码, 即用二进制数字或位来对字符串进行编码. 位串越短, 储存和传输信息也就越经济越容易.

例 11.5 对信息 ABAACDAB 进行二进制字符串编码.

由于里边有 4 个符号, 一种方便的二进位编码是用两位分别来表示各个字母. 例如, 我们可以建立如下相应联系:

A	00
B	01
C	10
D	11

然后这信息被编码成:

$$(00)(01)(00)(00)(10)(11)(00)(01)$$

使用这种编码, 我们总共需要 16 位来储存和传输信息. ◀

然而还有更有效的编码方式. 为便于理解, 我们先引入信息这个概念. 假设有 K 个不同的符号, 并且用 p_i 记第 i 种符号在串中任意点出现的概率. 该概率可能是事先知道的, 或者可能是依经验把第 i 种符号在串中出现的次数除以串的长度估计出来的.

定义 11.7 串的**香农信息(Shannon information)**或**香农熵(Shanon entropy)**是 $I=-\sum_{i=1}^{k} p_i \log_2 p_i$.

该定义是以贝尔实验室 (Bell Laboratories) 的 C. Shannon 名字命名的, 他在 20 世纪中叶做了信息论的开创性工作. 字符串的香农信息被认为是对信息进行编码时平均每个符号所需的最小位数. 逻辑是这样的: 在平均意义下, 如果一个符号以 p_i 的频率出现, 那么期望用 $-\log_2 p_i$ 位来表示它. 例如, 一个有 1/8 的机会出现的符号可以被 $-\log_2(1/8)=3$ 位符号 000, 001, $\cdots$, 111 之一来表示. 为找到所有符号的平均位数, 我们应该通过每个符号 i 出现的概率 p_i 作为它的权重. 这意味着对于整条信息而言, 每个位符号的平均数是定义中的和 I.

例 11.6 求字符串ABAACDAB的香农信息.

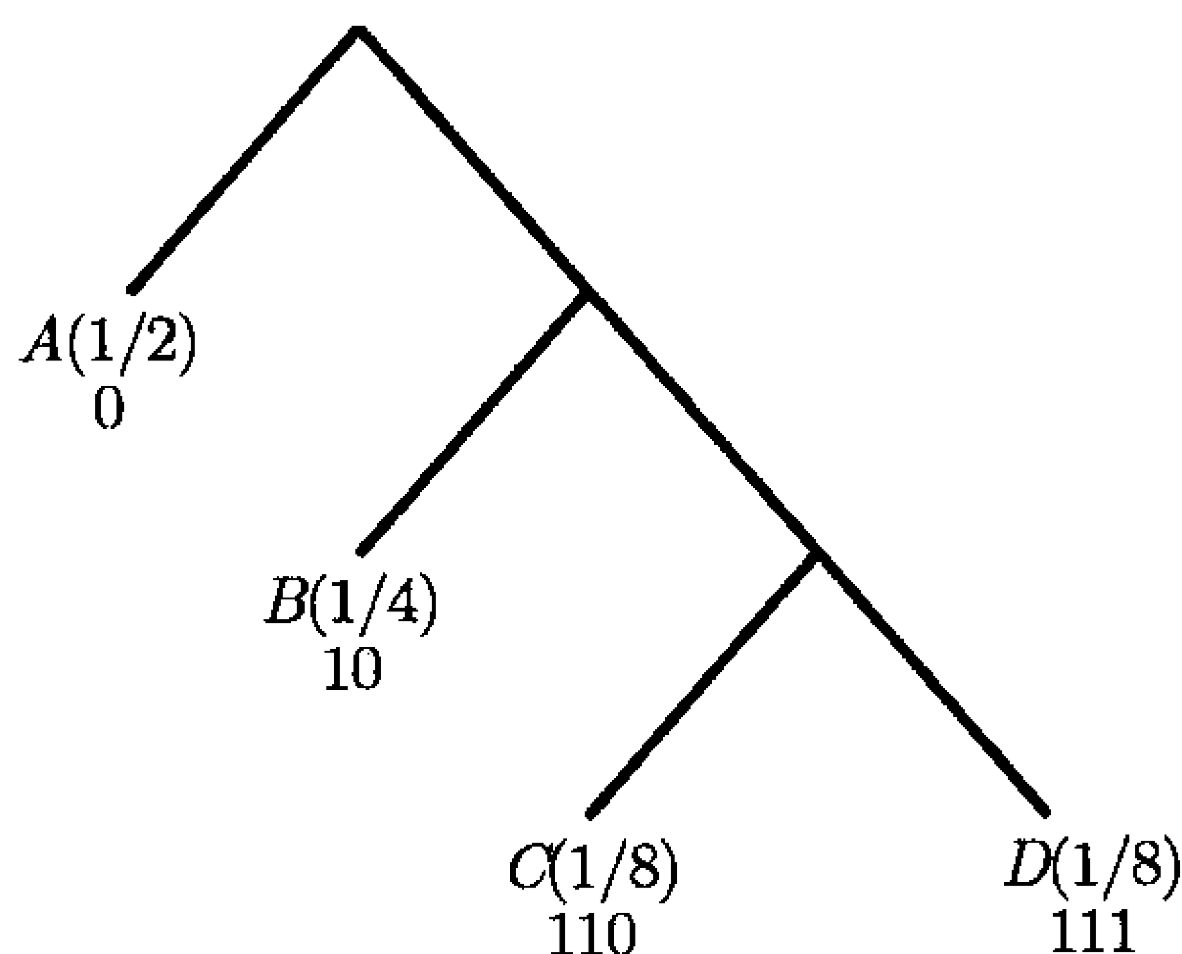
A, B, C, D 出现的经验概率分别为 $p_1=4/8=2^{-1}, p_2=2/8=2^{-2}$, $p_3=1/8=2^{-3}$, $p_4=2^{-3}$, 所以香农信息为

$$-\sum_{i=1}^{4} p_i \log_2 p_i=\frac{1}{2}\times 1+\frac{1}{4}\times 2+\frac{1}{8}\times 3+\frac{1}{8}\times 3=\frac{7}{4}.$$ ◀

因此, 由香农信息估计每个字符至少需要 1.75 位来对该字符串进行编码. 由于字符串的长度为 8, 用来储存信息的最优位数为 (1.75)(8)=14, 而不是像我们在前面编码所需的 16.

事实上, 使用 Huffman**编码**这种方法能使上述信息以我们前面所预测的 14 位来输送. 目标是赋给每个符号一个可以反映该符号出现概率的二进位数, 越常出现的符号码越短.

我们通过建立一个从中可以读取二进制码的树来使用该算法. 开始我们选取两个出现概率最小的符号, 然后考虑 "组合" 的符号, 赋给它们组合概率. 这两个符号组成了树的一枝. 然后重复上述步骤, 把符号进行组合并且使它们成为树枝, 直到只剩一个字符组, 也就是树的顶端. 在例 11.6 中, 我们首先两个出现概率最低的符号 C 和 D 进行组合并赋以概率 1/4. 剩下的概率分布为 A(1/2),B(1/4),CD(1/4). 再一次把两个出现可能性最小的符号进行组合有 A(1/2),BCD(1/2). 最后, 把剩下的两个进行组合得 ABCD(1). 每个组合组成了 Huffman 树的一枝:



一旦树成了完全树, 每个符号的 Huffman 编码可以从树的顶端往下寻找得到, 如上所示, 当往左枝时记为 0, 往右时记为 1. 例如, A 通过 0 来表示, C 通过两次向右一次向左 (110) 来表示. 现在例 11.6 中的字符串可以翻译为一个长度为 14 的位流:

$$(0)(10)(0)(0)(110)(111)(0)(10).$$

香农信息提供了二进位编码平均每个字符的位数的下界. 在例题所示的情况下, Huffman 编码达到香农信息界 $14/8 = 1.75$ 位/字符. 不巧的是, 如下例所示, 并不是经常可以做到的.

例 11.7 求字符串`ABRA CADABRA`的香农信息并对其进行 Huffman 编码.

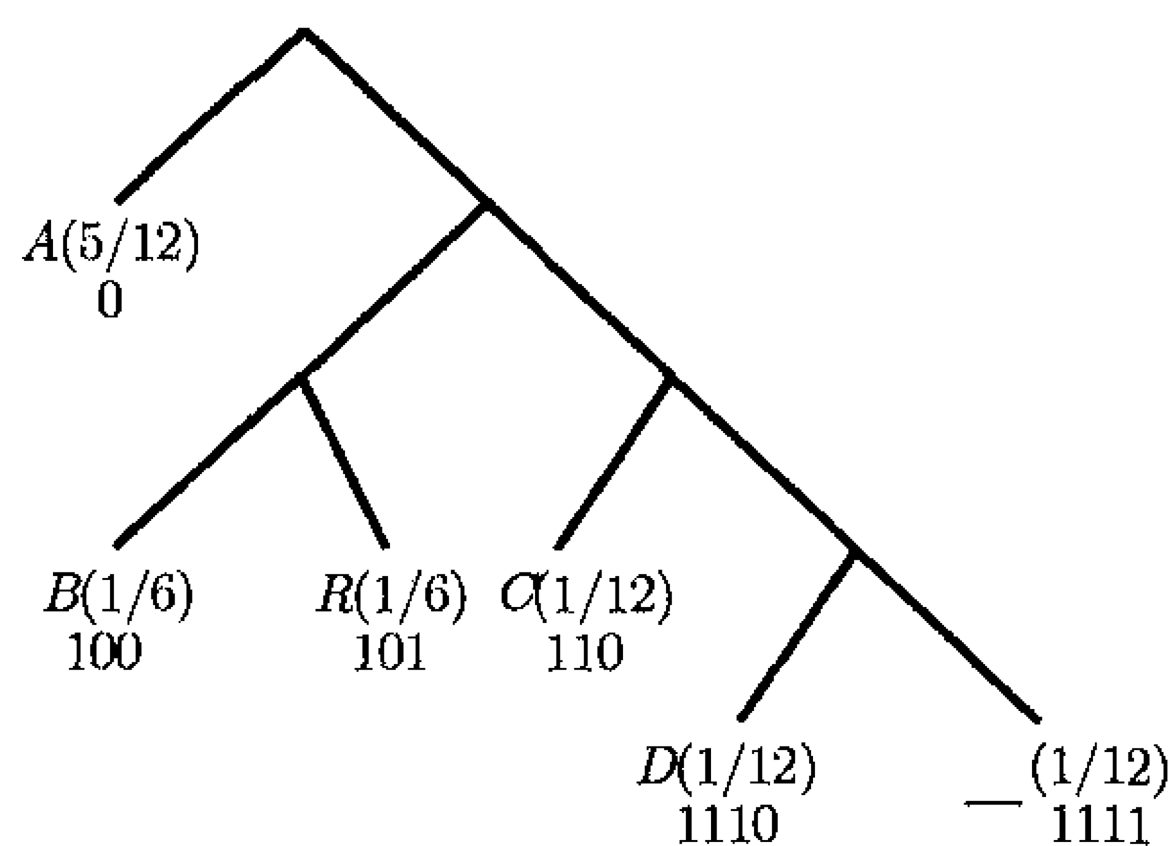
这 6 个符号的经验概率为

A	5/12
B	2/12
R	2/12
C	1/12
D	1/12
—	1/12

注意空格也作为一个符号. 香农信息为

$$-\sum_{i=1}^{6} p_i \log_2 p_i = -\frac{5}{12}\log_2\frac{5}{12} - 2\times\frac{1}{6}\log_2\frac{5}{6} - 3\times\frac{1}{12}\log_2\frac{1}{12} \approx 2.28 \text{ 位/字符}.$$

这是为ABRA CADABRA进行编码时每个符号所需最小位数的理论值. 为找到 Huffman 编码, 我们可以用前面介绍的过程进行处理. 尽管任意两个概率为 1/12 的字符都可以选作最低下的树枝, 但开始时我们把 D 和空格进行组合. 最后一个是 A, 因为它具有最大的概率. 一种 Huffman 编码如下图所示:



由于 A 在字符串中经常出现, 它的编码最短. ABRA CADABRA的二进位编码是一个长度为 28 位的序列:

$$(0)(100)(101)(0)(1111)(110)(0)(1110)(0)(100)(101)(0).$$

这个编码平均 $28/12 = 2\frac{1}{3}$ 位/字符, 比前面计算的理论最小值稍微大一点. Huffman 码并不总是和香农信息相匹配, 但是经常是很接近的. ◀

Huffman 编码的秘密之处在于, 字符仅在树枝的末端出现, 一个完整的字符编码不可能成为另一个字符编码的开始. 因此, 在把编码译回符号时不会出现任何含糊.

11.3.2 JPEG 格式的 Huffman 编码

本节专门介绍 Huffman 编码在实际应用中的一个拓展例子. JPEG 图像压缩格式在现代数字摄影中随处可见. 这成为理论数学和工程应用相结合的非常吸引人的例子.

用两种不同方式对 JPEG 图像文件变换系数进行二进制 Huffman 编码, 一种是 DC 部分 [变换矩阵的 (0,0) 元素], 另一种为 8×8 剩下的 63 个元素, 也就是所谓的 AC 分部.

定义 11.8 *设 y 是一个整数. y 的大小定义为*

$$L=\begin{cases}\mathrm{floor}(\log_2|y|)+1, & y\neq 0,\\ 0, & y=0.\end{cases}$$

JPEG 的 Huffman 编码有 3 个组成部分: 一个是 DC 部分的 Huffman 树, 另一个是 AC 部分的 Huffman 树, 还有一个就是整数识别表. 对于元素 $y=y_{00}$ 的第一部分编码是对 y 的大小进行二进制编码, 从下面的 DC 部分的 Huffman 树 (被称为 DPCM 树) 可以得到:

和前面一样, 当向树枝的左端或右端分支时分别记为 0 或 1. 第一部分是二进制串的整数识别表 11-1.

表 11-1

L	元 素	二 进 制
0	0	——
1	$-1,1$	$0,1$
2	$-3,-2,2,3$	$00,01,10,11$
3	$-7,-6,-5,-4,4,5,6,7$	$000,001,010,011,100,101,110,111$
4	$-15,-14,\cdots,-8,8,\cdots,14,15$	$0000,0001,\cdots,0111,1000,\cdots,1110,1111$
5	$-31,-30,\cdots,-16,16,\cdots,30,31$	$00000,00001,\cdots,01111,10000,\cdots,11110,11111$
6	$-63,-62,\cdots,-32,32,\cdots,62,63$	$000000,000001,\cdots,011111,100000,\cdots,111110,111111$
⋮	⋮	⋮

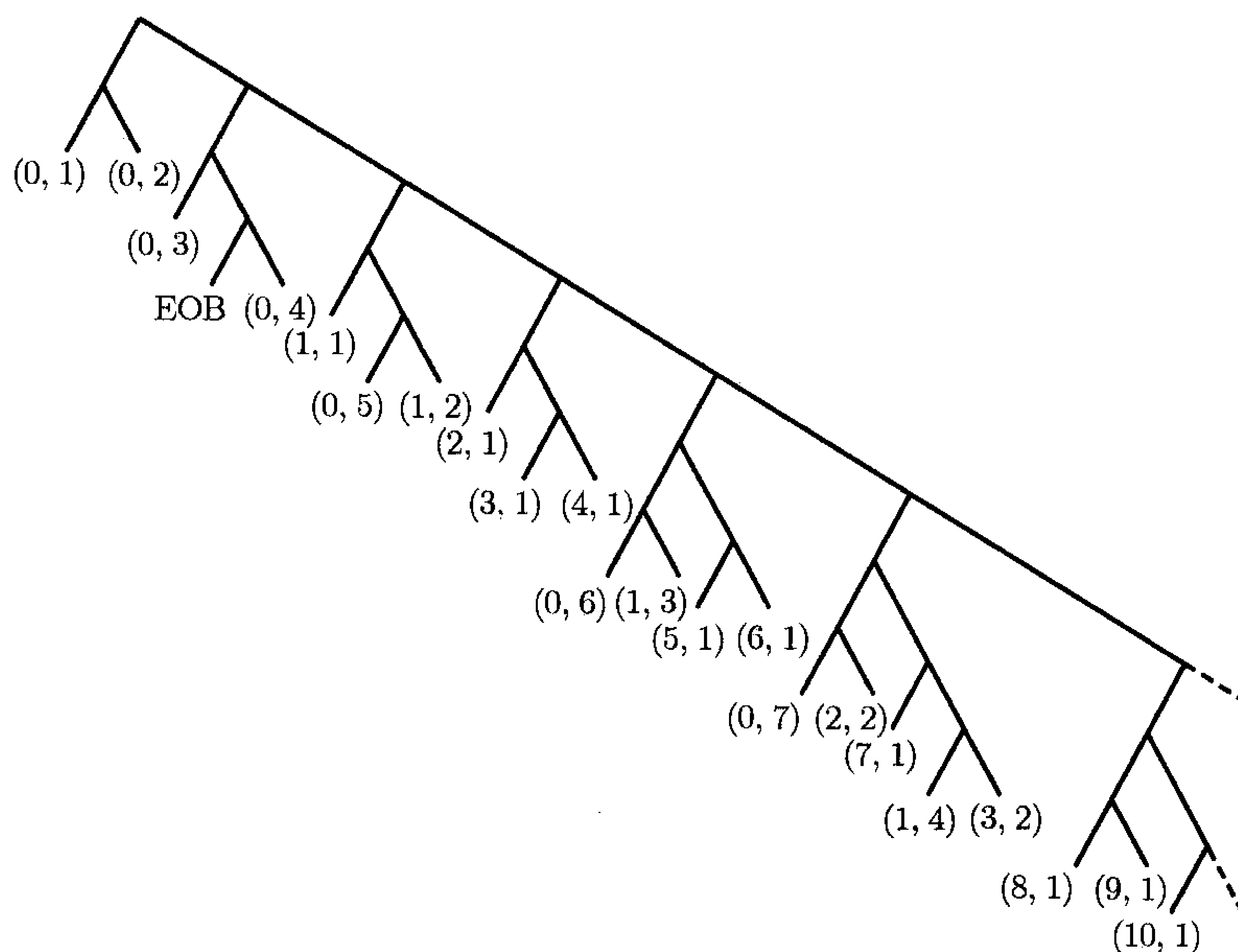
例如, 元素 $y_{00}=13$ 的大小 $L=4$. 根据 DPCM 树, 4 的 Huffman 编码是 (101). 该表也说明了 13 的二进制编码是 (1101), 所以连接这两部分的 1011101 可以用来储存 DC 系数.

因为 DC 部分和边上的 8×8 像素块常有联系, 我们仅储存块与块之间的差 (当然, 第一块除外). DC 部分从左向右的差像所示的那样存储. 这种方法被称为 DC 部分的差值脉冲编码调制 (DPCM).

对于 8×8 像素块的剩下的 63 个 AC 部分, 游程编码用于有效地存储很多的 0. 用来存储 63 个部分的惯用顺序是 "之" 字模式:

$$\begin{bmatrix} 0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{bmatrix}. \tag{11.26}$$

我们对一个零游程对 (n, L) 进行编码, 其中 n 代表一系列零的长度, L 代表下一个非零元素的大小, 而不是对这 63 个数字本身进行编码. 在典型的 JPEG 图像中经常遇到的编码和依 JPEG 标准的默认编码显示在 AC 部分的 Huffman 树中.



在位流中, 从前面的表中, 由树 (仅仅识别元素的大小) 中得到的 Huffman 码直接跟着识别该整数的二进制码. 例如, 元素序列 −5, 0, 0, 0, 2 将表示成 (0, 3)−5(3, 2)2, 其中 (0,3) 意思是编码长度为 3 的数之前没有 0, (3, 2) 意思是编码长度为 2

的数之前有 3 个 0. 从 Huffman 树中, 我们可以得到 (0, 3) 可以编成码 (100), (3, 2) 可以编成码 (111110111). 从整数识别表中可知, 识别 5 的是 (010), 识别 2 的是 (10). 因此 $-5, 0, 0, 0, 2$ 的位流为 (100)(010)(111110111) (10).

前面的 Huffman 树仅仅显示了 JPEG 中最经常碰到的游程编码. 另外一个有用的编码是 (11,1)=1111111001, (12,1)=1111111010, (13,1)=11111111000.

例 11.8 为 JPEG 图像文件对 (11.21) 中的量化 DCT 变换矩阵进行编码.

DC 元素 $y_{00} = -38$ 通过 DPCM 树编码为 (1110), 从整数识别表得二进位编码为 (011001), 其大小为 6. 接下来考虑 AC 系数串. 根据 (11.26), AC 系数如下排序: $13, -20, 4, -11, 4, -2, 2, -3, 3, 0, 1, -2, 2, 0, 0, 0, 0, 0, 1$, 剩下的全为 0. 开始, 13 的编码大小为 4, 所以我们为游程编码构造对 (0, 4) 和从识别表中得到 1101. 下一个数为 -20, 其编码长度为 5, 因此得 (0, 5) 和 01011. 所有的零游程对为

(0, 4)13(0, 5)−20(0, 3)4(0,4)−11(0,3)4(0,2)−2(0,2)2(0,2)−3

(0,2)3(1,1)1(0,2)−2(0,2)2(5,1)1(EOB)

这里的EOB表示块的结尾 (End of Block) 即剩下的元素全为 0. 根据 Huffman 树, EOB由码 (1010) 表示. 储存相片的某个 8×8 像素块的位流可以从 Huffman 树和整数识别表得到:

(1110)(011001)

(1011)(1101)(11010)(01011)(100)(100)(1011)(0100)(100)(100)(01)(01)

(01)(10)(01)(00)(01)(11)(1100)(1)(01)(01)(01)(10)(1111010)(1)(1010).

这 89 位二进制数精确地表示了图 11-8a 的像素块——它是图 11-6a 的原始图的很合理逼近. 平均每个像素占用 $89/64 \approx 1.4$ 位. 我们可以发觉这种编码比仅仅用低通过滤器和量化得到的每像素位数要优良得多. 由于开始存储每个整数要 8 位, 这 8×8 的图像压缩后只占开始空间的 1/5.

解压缩 JPEG 文件的过程包括了压缩过程的逆. ◀

JPEG 阅读器把位流解码成游程符号, 这组成了 8×8 的 DCT 变换程序块, 它们通过使用 DCT 的逆最后依次转换回像素块.

习题 11.3

1. 求解下列信息中每个符号出现的概率和香农信息:

   (a) BABBCABB;　(b) ABCACCAB;　(c) ABABCABA.

2. 画出习题 1 的 Huffman 树并对信息进行编码. 并将香农信息和编码中每个字符所需的平均位数进行比较.

3. 画出 Huffman 树并把包含空格和感叹号的信息通过 Huffman 编码转换成位流. 并将香农信息和编码中每个字符所需的平均位数进行比较.

   (a) AY CARUMBA!　(b) COMPRESS THIS MESSAGE　(c) SHE SELLS SEASHELLS BY THE SEASHORE

4. 使用 JPEG Huffman 编码, 将经转换和量化的矩阵 (a)(11.22) 和 (b)(11.23) 转换成位流.

11.4 改进的 DCT 和音频压缩

我们回到一维信号的问题, 并且讨论音频压缩的一些最新方法. 尽管我们可能会认为一维的信号要比二维好处理一些, 问题在于人类听觉系统在频域内很敏感, 通过压缩和解压缩所引入的噪音很容易被发觉. 因此, 在声音压缩中, 通常使用复杂的技巧以隐藏声音被压缩过的事实.

我们首先介绍一种新改进的 DCT——DCT4, 也就是所谓的改进的离散余弦变换 (MDCT). MDCT 由一个不是方阵的矩阵表示, 因此不像 DCT, DCT4 是不可逆的. 尽管如此, 当应用于重叠的窗口时, 它可以完全重构原始的数据流. 更重要的是, 它可以和量化一起执行有损压缩使得带来的声音质量失真最小. 这种 MDCT 是当前很多音频压缩文件格式的核心, 如: MP3,AAC, WMA.

11.4.1 MDCT

我们先介绍和前面讲的在形式上略微不同的 DCT. 经常使用的 DCT 有 4 个不同的形式. 在前面我们使用的 DCT1 文件用来压缩影像, 而文件 DCT4 经常被用来压缩声音.

定义 11.9(DCT4) *$\boldsymbol{x}=(x_0,\cdots,x_{n-1})^{\mathrm{T}}$ 的 DCT4 是 n 维向量: $\boldsymbol{y}=\boldsymbol{E}\boldsymbol{x}$, 其中 $\boldsymbol{E}$ 是 $n\times n$ 的矩阵*

$$E_{ij}=\sqrt{\frac{2}{n}}\cos\frac{(i+\frac{1}{2})(j+\frac{1}{2})\pi}{n} \quad . \tag{11.27}$$

像 DCT1 一样, DCT4 中的矩阵 $\boldsymbol{E}$ 是实的正交阵: 它是方阵并且它的列是正交单位向量. 后者是由于 $\boldsymbol{E}$ 的列是 $n\times n$ 实对称阵

$$\begin{pmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 3 \end{pmatrix} \tag{11.28}$$

的单位特征向量. 习题 6 要求读者验证这一事实.

下面来看 DCT4 矩阵列向量的两个重要事实. 假设 n 是固定的, 不仅要考虑 DCT4 矩阵中的 n 列, 还要考虑由 (11.27) 定义的 j 取所有的正整数和负整数的列向量.

引理 11.10 以 $\boldsymbol{c}_j$ 记 (11.27) 中 (延拓的) DCT4 矩阵的第 j 列. 对于所有的整数 j 有 (a) $\boldsymbol{c}_j = \boldsymbol{c}_{-1-j}$(列关于 $j=-1/2$ 对称) 和 (b)$\boldsymbol{c}_j = -\boldsymbol{c}_{2n-1-j}$(列关于 $j = n-1/2$ 反对称).

证 为证明引理中的 (a), 把 j 写成 $-1/2+(j+1/2)$, 把 $-1-j$ 写成 $-1/2-(j+1/2)$. 使用 (11.27) 定义, 对 $i=1,\cdots,n-1$ 有

$$
\begin{aligned}
\boldsymbol{c}_j = \boldsymbol{c}_{-\frac{1}{2}+\left(j+\frac{1}{2}\right)} &= \sqrt{\frac{2}{n}}\cos\frac{\left(i+\frac{1}{2}\right)\left(j+\frac{1}{2}\right)\pi}{n} = \sqrt{\frac{2}{n}}\cos\frac{\left(i+\frac{1}{2}\right)\left(-j-\frac{1}{2}\right)\pi}{n}\\
&= \boldsymbol{c}_{-\frac{1}{2}-\left(j+\frac{1}{2}\right)} = \boldsymbol{c}_{-1-j}
\end{aligned}
$$

为证明 (b), 设 $r=n-1/2-j$. 有 $j=n-1/2-r$ 和 $2n-1-j=n-1/2+r$, 我们必须证明 $\boldsymbol{c}_{n-1/2-r}+\boldsymbol{c}_{n-1/2+r}=0$. 通过余弦加法公式, 对 $i=1,\cdots,n-1$ 有

$$
\begin{aligned}
\boldsymbol{c}_{n-\frac{1}{2}-r} =& \sqrt{\frac{2}{n}}\cos\frac{(2i+1)(n-r)\pi}{2n} = \sqrt{\frac{2}{n}}\cos\frac{2i+1}{2}\pi\cos\frac{(2i+1)r\pi}{2n}\\
&+\sqrt{\frac{2}{n}}\sin\frac{2i+1}{2}\pi\sin\frac{(2i+1)r\pi}{2n}\\
\boldsymbol{c}_{n-\frac{1}{2}+r} =& \sqrt{\frac{2}{n}}\cos\frac{(2i+1)(n+r)\pi}{2n} = \sqrt{\frac{2}{n}}\cos\frac{2i+1}{2}\pi\cos\frac{(2i+1)r\pi}{2n}\\
&-\sqrt{\frac{2}{n}}\sin\frac{2i+1}{2}\pi\sin\frac{(2i+1)r\pi}{2n}.
\end{aligned}
$$

由于对于任意的整数 i,$\cos\frac{1}{2}(2i+1)\pi=0$, 则得 $\boldsymbol{c}_{n-1/2-r}+\boldsymbol{c}_{n-1/2+r}=0$.

我们将使用 DCT4 矩阵 $\boldsymbol{E}$ 来建立 MDCT. 假设 n 是偶数. 我们将使用列 $\boldsymbol{c}_{\frac{n}{2}},\cdots,\boldsymbol{c}_{\frac{5n}{2}-1}$ 建立一个新的矩阵. 引理 11.10 证明了对于任意整数 j, $\boldsymbol{c}_j$ 列可以表为 DCT4 的一列. 如图 11-10 所示, 对于 $0\leqslant i\leqslant n-1$ 的 $\boldsymbol{c}_i$, 只是可能要做一种符号改变.

$\cdots\ c_{-4}\ c_{-3}\ c_{-2}\ c_{-1}\ c_0\ c_1\ c_2\ \cdots\ \cdots\ c_{n-1}\ c_n\ \cdots\ \cdots\ c_{2n-1}\ c_{2n}\ c_{2n+1}\ \cdots$

$\cdots\ c_3\ c_2\ c_1\ c_0\ c_0\ c_1\ c_2\ \cdots\ \cdots\ c_{n-1}\ -c_{n-1}\ \cdots\ \cdots\ -c_0\ -c_0\ -c_1\ \cdots$

图 11-10 引理 11.10 的描述: $\boldsymbol{c}_0,\cdots,\boldsymbol{c}_{n-1}$ 列向量组成了 $n\times n$ 的 DCT4 矩阵. 根据引理 11.10, 对于这范围外的 j, 由 (11.27) 式 $\boldsymbol{c}_j$ 所定义的列和 DCT4 中的某列相当

定义 11.11 设 n 为正偶数, $\boldsymbol{x}=(x_0,\cdots,x_{n-1})^{\mathrm{T}}$ 的 MDCT 是 n 维向量

$$\boldsymbol{y}=\boldsymbol{M}\boldsymbol{x} \tag{11.29}$$

其中 M 为 $n \times 2n$ 的矩阵

$$M_{ij} = \sqrt{\frac{2}{n}} \cos \frac{(i + \frac{1}{2})(j + \frac{n}{2} + \frac{1}{2})\pi}{n}, \quad 0 \leqslant i \leqslant n-1, 0 \leqslant j \leqslant 2n-1. \tag{11.30}$$

和前面介绍的 DCT 形式的最主要不同之处在于: 长度为 $2n$ 的向量的 MDCT 是长度为 n 的向量. 因此, MDCT 不是直接可逆的, 但是通过交叠长度为 $2n$ 的向量, 我们可以达到同样的效果.

和定义 11.9 相比, 可以把 MDCT 矩阵 M 通过 DCT4 的列写出来, 然后使用引理 11.10 进行化简:

$$\begin{aligned}\boldsymbol{M} &= \left[\boldsymbol{c}_{\frac{n}{2}} \cdots \boldsymbol{c}_{\frac{5}{2}n-1}\right] = \left[\boldsymbol{c}_{\frac{n}{2}} \cdots \boldsymbol{c}_{n-1} \middle| \boldsymbol{c}_n \cdots \boldsymbol{c}_{\frac{3}{2}n-1} \middle| \boldsymbol{c}_{\frac{3}{2}n} \cdots \boldsymbol{c}_{2n-1} \middle| \boldsymbol{c}_{2n} \cdots \boldsymbol{c}_{\frac{5}{2}n-1}\right] \\ &= \left[\boldsymbol{c}_{\frac{n}{2}} \cdots \boldsymbol{c}_{n-1} \middle| -\boldsymbol{c}_{n-1} \cdots -\boldsymbol{c}_{\frac{n}{2}} \middle| -\boldsymbol{c}_{\frac{1}{2}n-1} \cdots -\boldsymbol{c}_0 \middle| -\boldsymbol{c}_0 \cdots -\boldsymbol{c}_{\frac{n}{2}n-1}\right].\end{aligned} \tag{11.31}$$

为简化记号, 记 DCT4 矩阵的左半部分和右半部分分别为 $\boldsymbol{A}$ 和 $\boldsymbol{B}$, 即 $\boldsymbol{E} = (\boldsymbol{A}|\boldsymbol{B})$. 从左到右交换单位矩阵的列定义, 得到的置换矩阵为

$$\boldsymbol{R} = \begin{pmatrix} & & 1 \\ & \cdot^{\cdot^{\cdot}} & \\ 1 & & \end{pmatrix}.$$

当置换矩阵 $\boldsymbol{R}$ 乘在矩阵的右边时, 它把矩阵的右边的列换到了左边; 当乘在左边, 它把矩阵的下边的行换到了上边. 由于 $\boldsymbol{R}$ 的逆等于 $\boldsymbol{R}$ 的转置, 也等于 $\boldsymbol{R}$ 本身, 我们知道 $\boldsymbol{R}$ 为对称正交阵. 现在 (11.31) 可以更简单地写成

$$\boldsymbol{M} = (\boldsymbol{B}| -\boldsymbol{BR}| -\boldsymbol{AR}| -\boldsymbol{A}), \tag{11.32}$$

其中 $\boldsymbol{AR}$ 和 $\boldsymbol{BR}$ 是 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的列经左右置换得到的.

MDCT 的作用可以通过 DCT4 表示出来. 令

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \end{pmatrix}$$

是一个 $2n$ 维向量, 其中每个 $\boldsymbol{x}_i$ 是长度为 $n/2$ 的向量 (记住 n 是偶数). 通过 (11.32) 中 $\boldsymbol{M}$ 的特征化, 有

$$\boldsymbol{Mx} = \boldsymbol{Bx}_1 - \boldsymbol{BRx}_2 - \boldsymbol{ARx}_3 - \boldsymbol{Ax}_4 = [\boldsymbol{A}|\boldsymbol{B}] \begin{pmatrix} -\boldsymbol{Rx}_3 - \boldsymbol{x}_4 \\ \boldsymbol{x}_1 - \boldsymbol{Rx}_2 \end{pmatrix} = \boldsymbol{E} \begin{pmatrix} -\boldsymbol{Rx}_3 - \boldsymbol{x}_4 \\ \boldsymbol{x}_1 - \boldsymbol{Rx}_2 \end{pmatrix}, \tag{11.33}$$

其中 $\boldsymbol{E}$ 是 $n \times n$ DCT4 的矩阵, $\boldsymbol{Rx}_2$ 和 $\boldsymbol{Rx}_3$ 表示 $\boldsymbol{x}_2$ 和 $\boldsymbol{x}_3$ 的元素进行了上下置换. 这样做非常有用, 我们可以把 $\boldsymbol{M}$ 的输出结果用正交阵 $\boldsymbol{E}$ 表示出来.

由于 $n\times 2n$ 的 MDCT 矩阵 $\boldsymbol{M}$ 不是一个方阵, 它不是可逆的. 尽管如此, 相邻的两个 MDCT 矩阵的秩和可以为 $2n$, 并且一起作用可以完全重构 $\boldsymbol{x}$ 的输入值.

MDCT 的 "逆" 可以表示为 $2n \times n$ 的矩阵 $\boldsymbol{N} = \boldsymbol{M}^{\mathrm{T}}$, 它的转置矩阵的元素是

$$N_{ij} = \sqrt{\frac{2}{n}} \cos \frac{(j+\frac{1}{2})(i+\frac{n}{2}+\frac{1}{2})\pi}{n}. \tag{11.34}$$

尽管它和一个矩形矩阵很接近, 但它不是事实上的逆. 通过对 (11.32) 进行转置, 使用前面的符号 $\boldsymbol{E} = (\boldsymbol{A}|\boldsymbol{B})$ 来记 DCT4, 我们有

$$\boldsymbol{N} = \begin{bmatrix} \boldsymbol{B}^{\mathrm{T}} \\ -\boldsymbol{RB}^{\mathrm{T}} \\ -\boldsymbol{RA}^{\mathrm{T}} \\ -\boldsymbol{A}^{\mathrm{T}} \end{bmatrix}. \tag{11.35}$$

由于 $\boldsymbol{E}$ 为一个正交阵, 我们得到

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I},$$

$$\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \boldsymbol{I},$$

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{0},$$

其中 $\boldsymbol{I}$ 表示 $n \times n$ 的单位阵.

现在我们计算 $\boldsymbol{NM}$, 看在什么意义下 $\boldsymbol{N}$ 施逆于 MDCT 矩阵 $\boldsymbol{M}$. $\boldsymbol{x}$ 像前面一样分为 4 部分. 根据 (11.33) 和 (11.35)、$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的正交性, 以及事实 $\boldsymbol{R}^2 = \boldsymbol{I}$, 我们有

$$\boldsymbol{NM}\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}^{\mathrm{T}} \\ -\boldsymbol{RB}^{\mathrm{T}} \\ -\boldsymbol{RA}^{\mathrm{T}} \\ -\boldsymbol{A}^{\mathrm{T}} \end{bmatrix} [\boldsymbol{A}(-\boldsymbol{Rx}_3 - \boldsymbol{x}_4) + \boldsymbol{B}(\boldsymbol{x}_1 - \boldsymbol{Rx}_2)] = \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{Rx}_2 \\ -\boldsymbol{Rx}_1 + \boldsymbol{x}_2 \\ \boldsymbol{x}_3 + \boldsymbol{Rx}_4 \\ \boldsymbol{Rx}_3 + \boldsymbol{x}_4 \end{bmatrix} \tag{11.36}$$

在音频压缩算法中, 我们把 MDCT 应用到折叠的数据向量. 由于向量的长度是个常数, 由于向量尾部产生的人为噪音将会以固定的频率发生. 听觉系统比视觉系统对周期性的误差甚至更敏感; 毕竟, 一种固定频率的误差是那种频率的音调,

是耳朵可以感觉到的. 假设数据通过折叠的方式表示. 令

$$Z_1 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \text{ 和 } Z_2 = \begin{bmatrix} x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

为两个 $2n$ 维向量, 其中 n 为偶数, 每个 x_i 为长度为 $n/2$ 的向量. 向量 Z_1 和 Z_2 折叠了它们长度的一半. 由于 (11.36) 证明了

$$NMZ_1 = \begin{bmatrix} x_1 - Rx_2 \\ -Rx_1 + x_2 \\ x_3 + Rx_4 \\ Rx_3 + x_4 \end{bmatrix} \text{ 和 } NMZ_2 = \begin{bmatrix} x_3 - Rx_4 \\ -Rx_3 + x_4 \\ x_5 + Rx_6 \\ Rx_5 + x_6 \end{bmatrix}, \tag{11.37}$$

我们可以通过平均 NMZ_1 的下半部和 NMZ_2 的上半部来精确地重构 n 维向量 $[x_3, x_4]$:

$$\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \frac{1}{2}(NMZ_1)_{n,\cdots,2n-1} + \frac{1}{2}(NMZ_2)_{0,\cdots,n-1}. \tag{11.38}$$

这个等式介绍了 N 如何用于对经 M 编码后的信号进行解码.

这个结果在定理 11.12 中进行了概括.

定理 11.12(折叠后 MDCT 的逆) *设 M 为 $n\times 2n$ 的 MDCT 矩阵, $N = M^{\mathrm{T}} \cdot u_1, u_2, u_3$ 为 n 维向量, 又设*

$$u_1 = M \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \text{ 和 } v_2 = M \begin{bmatrix} u_2 \\ u_3 \end{bmatrix}.$$

那么通过

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = Nv_1 \text{ 和 } \begin{bmatrix} w_3 \\ w_4 \end{bmatrix} = Nv_2$$

定义的 n 维向量 w_1, w_2, w_3, w_4 满足 $u_2 = (w_2 + w_3)/2$.

这是精确的重构. 定理 11.12 经常和连接起来的 n 维向量 $[u_1, u_2, \cdots, u_{m-1}]$ 的长信号一起使用. MDCT 应用于连接对以得到转换后的信号 $(v_1, v_2, \cdots, v_{m-1})$. 现在的有损压缩来了. v_i 是频率部分, 因此可以选择保留某些频率而忽略另外一些. 11.4.2 节将继续这样做.

通过量化或其他一些手段, v_i 的内容缩减了, $(u_2, \cdots, u_{m-1})$ 可以通过定理 11.12 被压缩. 可以看到 u_1 和 u_m 不可能恢复; 它们应该是信号的不重要部分或者是先前人为加进去的.

例 11.9 利用折叠的 MDCT 对信号 $\boldsymbol{x}=[1, 2, 3, 4, 5, 6]$ 进行转换. 然后使用逆变换重构中间部分 [3, 4].

我们将折叠向量 [1,2,3,4] 和 [3,4,5,6]. 设 $n=2$ 并置

$$\boldsymbol{E}_2=\begin{bmatrix}\cos\frac{\pi}{8} & \cos\frac{3\pi}{8}\\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8}\end{bmatrix}=\begin{bmatrix}b & c\\ c & -b\end{bmatrix}.$$

应用 2×4 的 MDCT 得

$$\boldsymbol{v}_1=\boldsymbol{M}\begin{bmatrix}1\\2\\3\\4\end{bmatrix}=\boldsymbol{E}_2\begin{bmatrix}-\boldsymbol{R}(3)-4\\ 1-\boldsymbol{R}(2)\end{bmatrix}=\boldsymbol{E}_2\begin{bmatrix}-7\\-1\end{bmatrix}=\begin{bmatrix}-7b-c\\ b-7c\end{bmatrix}=\begin{bmatrix}-6.849\,8\\ -1.754\,9\end{bmatrix},$$

$$\boldsymbol{v}_2=\boldsymbol{M}\begin{bmatrix}3\\4\\5\\6\end{bmatrix}=\boldsymbol{E}_2\begin{bmatrix}-\boldsymbol{R}(5)-6\\ 3-\boldsymbol{R}(4)\end{bmatrix}=\boldsymbol{E}_2\begin{bmatrix}-11\\-1\end{bmatrix}=\begin{bmatrix}-11b-c\\ b-11c\end{bmatrix}=\begin{bmatrix}-10.545\,4\\ -3.285\,6\end{bmatrix}.$$

转换后的信号为

$$[\boldsymbol{v}_1|\boldsymbol{v}_2]=\begin{bmatrix}-6.849\,8 & -10.545\,4\\ -1.754\,9 & -3.285\,6\end{bmatrix}.$$

为了对 MDCT 进行逆变换, 定义 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 为

$$\boldsymbol{E}_2=[\boldsymbol{A}\mid\boldsymbol{B}]=\left[\begin{array}{c|c}b & c\\ \\ c & -b\end{array}\right],$$

并计算

$$\begin{bmatrix}\boldsymbol{w}_1\\ \boldsymbol{w}_2\end{bmatrix}=\boldsymbol{N}\boldsymbol{v}_1=\begin{bmatrix}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{v}_1\\ -\boldsymbol{R}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{v}_1\\ -\boldsymbol{R}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v}_1\\ -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v}_1\end{bmatrix}=\begin{bmatrix}c & -b\\ -c & b\\ -b & -c\\ -b & -c\end{bmatrix}\begin{bmatrix}-7b-c\\ b-7c\end{bmatrix}=\begin{bmatrix}-1\\1\\7\\7\end{bmatrix},$$

$$\begin{bmatrix}\boldsymbol{w}_3\\ \boldsymbol{w}_4\end{bmatrix}=\boldsymbol{N}\boldsymbol{v}_2=\begin{bmatrix}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{v}_2\\ -\boldsymbol{R}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{v}_2\\ -\boldsymbol{R}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v}_2\\ -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{v}_2\end{bmatrix}=\begin{bmatrix}c & -b\\ -c & b\\ -b & -c\\ -b & -c\end{bmatrix}\begin{bmatrix}-11b-c\\ b-11c\end{bmatrix}=\begin{bmatrix}-1\\1\\11\\11\end{bmatrix},$$

其中我们已经用了 $b^2+c^2=1$ 这个事实. 借助于定理 11.12 的结果, 我们可以通过

$$\boldsymbol{u}_2=\frac{1}{2}(\boldsymbol{w}_2+\boldsymbol{w}_3)=\frac{1}{2}\left(\begin{bmatrix}7\\7\end{bmatrix}+\begin{bmatrix}-1\\1\end{bmatrix}\right)=\begin{bmatrix}3\\4\end{bmatrix}$$

重构折叠的部分 [3,4]. ◀

MDCT 的定义和使用不像本章前面所介绍的 DCT 那么直接. 它的好处在于允许以有效的方式折叠相邻的向量. 结果是从两个向量来平均影响, 减少在边缘可见的突然过渡而导致的人为噪音. 在 DCT 的例子中, 我们可以在重构信号前过滤或量化变换系数以改进或压缩信号. 接下来, 我们将介绍加上量化一步后 MDCT 如何用于压缩.

11.4.2 位的量化

音频信号的有损压缩通过对 MDCT 输出的信号进行量化来实现. 本节将对用于图像压缩的量化进行扩展, 允许对用来表示有损信号的位数有更多的控制.

从实数轴上的开区间 $(-L,L)$ 开始, 见图 11-11. 假设目的是想用 b 位表示 $(-L,L)$ 上的一个数, 并且容许很小的误差. 我们用 1 位做标记且量化成 $b-1$ 位的二进制数. 公式如下.

$(-L,L)$的b 位量化

量化: $z=\text{round}(y/q)$, 其中 $q=2L/(2^b-1)$

去量化: $\bar{y}=qz$ (11.39)

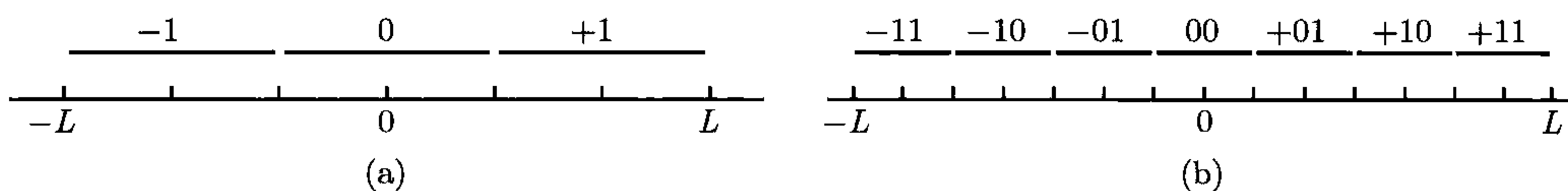


图 11-11 位量化. (11.39) 式的说明. (a)2 位; (b)3 位

作为一个例子, 我们说明如何用 4 位来表示区间 $(-1,1)$ 上的数字. 令 $q=2(1)/(2^4-1)=2/15$, 用 q 量化. 数 $y=-0.3$ 由

$$\frac{-0.3}{2/15}=-\frac{9}{4}\rightarrow -2\rightarrow -010$$

来表示, 数 $y=0.9$ 由

$$\frac{0.9}{2/15}=\frac{27}{4}=6.75\rightarrow 7\rightarrow +111$$

表示.

去量化是上面过程的逆. -0.3 的量化版本去量化后为

$$(-2)q=(-2)(2/15)=-4/15\approx -0.266\ 7,$$

而 0.9 的量化去量化后为

$$(7)q=(7)(2/15)=14/15\approx 0.933\ 3.$$

两种情况下, 量化误差都为 1/30.

例 11.10　用 4 位整数量化例 11.9 的 MDCT 输出. 然后去量化, 反转 MDCT 且算出量化误差.

变换矩阵的所有元素位于区间 $(-12,12)$. 用 $L=12$, 4 位量化要求 $q=2(12)/\left(2^4-1\right)=1.6$. 则

$$\boldsymbol{v}_1=\begin{bmatrix}-6.849\ 8\\-1.754\ 9\end{bmatrix}\to\begin{bmatrix}\text{round}\left(\frac{-6.894\ 8}{1.6}\right)\\\text{round}\left(\frac{-1.754\ 9}{1.6}\right)\end{bmatrix}\to\begin{bmatrix}-4\\-1\end{bmatrix}\to\begin{matrix}-100\\-001\end{matrix},$$

$$\boldsymbol{v}_2=\begin{bmatrix}-10.545\ 4\\-3.285\ 6\end{bmatrix}\to\begin{bmatrix}\text{round}\left(\frac{-10.545\ 4}{1.6}\right)\\\text{round}\left(\frac{-3.285\ 6}{1.6}\right)\end{bmatrix}\to\begin{bmatrix}-7\\-2\end{bmatrix}\to\begin{matrix}-111\\-010\end{matrix}.$$

变换的变量 $\boldsymbol{v}_1,\boldsymbol{v}_2$ 可以用 4 个 4 位整数存储, 共计 16 位.

用 $q=1.6$ 去量化为

$$\begin{bmatrix}-4\\-1\end{bmatrix}\to\begin{bmatrix}-6.4\\-1.6\end{bmatrix}=\bar{v}_1,\qquad\begin{bmatrix}-7\\-2\end{bmatrix}\to\begin{bmatrix}-11.2\\-3.2\end{bmatrix}=\bar{v}_2.$$

用 MDCT 的逆得到

$$\begin{bmatrix}\boldsymbol{w}_1\\\boldsymbol{w}_2\end{bmatrix}=\boldsymbol{N}\bar{\boldsymbol{v}}_1=\begin{bmatrix}-0.971\ 0\\0.971\ 0\\6.525\ 1\\6.525\ 1\end{bmatrix},\qquad\begin{bmatrix}\boldsymbol{w}_3\\\boldsymbol{w}_4\end{bmatrix}=\boldsymbol{N}\bar{\boldsymbol{v}}_2=\begin{bmatrix}-1.329\ 6\\1.329\ 6\\11.572\ 0\\11.572\ 0\end{bmatrix},$$

以及重构的信号

$$\boldsymbol{u}_2=\frac{1}{2}(\boldsymbol{w}_2+\boldsymbol{w}_3)=\frac{1}{2}\left(\begin{bmatrix}6.525\ 1\\6.525\ 1\end{bmatrix}+\begin{bmatrix}-1.329\ 6\\1.329\ 6\end{bmatrix}\right)=\begin{bmatrix}2.597\ 7\\3.927\ 4\end{bmatrix}.$$

量化误差是初始信号和重建信号的差:

$$\left\| \begin{bmatrix} 2.5977 \\ 3.9274 \end{bmatrix} - \begin{bmatrix} 3 \\ 4 \end{bmatrix} \right\| = \begin{bmatrix} 0.4023 \\ 0.0726 \end{bmatrix}.$$

◀

对音频文件编码通常是通过对规定的频率范围预先分配位数进行. 实例检验 11 指导读者如何构建一个使用 MDCT 及位量化的完整**编码译码器**(codec), 或者说是编码–解码草案.

习题 11.4

1. 求下列输入的 MDCT. 把结果用 $b=\cos\pi/8$ 和 $c=\cos 3\pi/8$ 表示.
   (a) [1,3,5,7] (b) [−2,−1,1,2] (c) [4,−1,3,5]
2. 像例 11.9 那样, 对于给定的输入求折叠窗口长度为 4 的 MDCT. 然后用 MDCT 的逆重构中间的部分.
   (a) [−3,−2,−1,1,2,3] (b) [1,−2,2,−1,3,0] (c) [4,1,−2,−3,0,3]
3. 将 (−1,1) 上的每个实数量化为 4 位, 然后去量化并计算量化误差.
   (a) 2/3 (b) 0.6 (c) 3/7
4. 重复练习 3, 但量化为 8 位.
5. 将 (−4,4) 上的每个实数量化为 8 位, 然后去量化并计算量化误差.
   (a) 3/2 (b) −7/5 (c) 2.9 (d) π
6. 证明对每个偶数 n, $n\times n$ 的 DCT4 矩阵是一个正交矩阵.
7. 在 (−6,6) 中量化到 4 位后, 重构练习 2 中数据的中间部分. 和准确的中间部分比较.
8. 在 (−6,6) 中量化到 6 位后, 重建练习 2 中数据的中间部分. 和准确的中间部分比较.
9. 解释为什么对任意的 k 按 (11.27) 定义的 n 维列向量 c_k 可以表示成列向量 $c_{k'}$, 其中 $0\leqslant k'\leqslant n-1$. 用这种方式表示 c_{5n} 和 c_{6n}.
10. 将一个实数转化为区间 $(-L,L)$ 的 b 位数时, 找出量化误差的一个上界 (由量化引起误差, 在去量化中仍有)

计算机问题 11.4

1. 写一个可以输入向量的 MATLAB程序, 将 MDCT 用于每个长度为 $2n$ 的窗口, 且像例 11.9 那样重构折叠长度为 n 的部分. 用下面这些输入信号进行试验.
   (a) n=4, $\boldsymbol{x}$=[1 2 3 4 5 6 7 8 9 10 11 12]
   (b) $n=4$, $x_i=\cos(i\pi/6), i=0,\cdots,11$
   (c) $n=8$, $x_i=\cos(i\pi/10), i=0,\cdots,63$
2. 修改你在计算问题 1 中的程序, 在重构折叠前加进 b 位量化. 然后重构问题中的例子且通过和原始输入比较来计算重建误差.

实例检验 11　把一个简单自动码用于 MDCT

音频文件的传输和存储是现代通讯的一个关键部分, 并且压缩在其中扮演相当重要的角色. 在这个实例检验中, 你将可以基于 MDCT 的能力, 将音频信号分解成频率部分和 11.4.2 节介绍的位量化方法, 建立起一个压缩–解压缩草案的骨架.

MDCT 应用于一个输入窗口为 $2n$ 的信号值并输出逼近数据的 n 个频率分量 (并且和下一个窗口一起对后面 n 个输入点进行插值). 算法的压缩部分包含了为节省存储空间的量化后的频率部分的编码, 如例 11.10 所示.

在通常的音频存储格式里, 在量化过程中对各种频率部分的位的分配基于心理声学 (研究人类感知声音的科学). 如频率掩模 (在给定时间里人耳只能接受每个频率段里主要声音的经验事实) 等技术被用来处理哪些频率是最需要和最不需要保留的. 更多的量化位分配给更重要的部分. 最具竞争性的方法是基于 MDCT 的并且在处理心理声学因素方面有差异. 在下面描述中, 我们将采用一个只依赖于重要性过滤的 (倾向于为大尺度频率部分分配更多的位) 忽略很多心理声学因素的简化方法.

我们开始介绍单音质声音的重构. 设 $t = 32$, 通过 MDCT 分类的最低下频率为 64Hz, 也是人耳所能感受到的最小的频率. 一个 64Hz 的音调由 $x(t) = \cos 2\pi(64)t$ 来表示, 其中 t 是通过秒来衡量的. 如果 F_s 是每秒的样本数, 那么 $1/F_s, 2/F_s, \cdots, F_s/F_s$ 表示一秒里的时间步. MATLAB命令

```
Fs=8192;
x=cos(2*pi*64*(1:Fs)/Fs);
sound(x,Fs)
```

表示了 64Hz 音的一秒. 值为 $8\,192 = 2^{13}$ 字节/秒的样本频率 Fs 非常常见, 相应地为 $2^{16} = 65\,536$ 位/秒, 对于音频文件为 64Kb/s 样本率.

以 64 的整数倍 $64f$ 代替 64 可以得到更高调的声音, 设 $f = 2$ 或 4 得到更高的八度音阶. 设 $f = 7$ 则得到 448Hz 的声音, 它和标准音 (440Hz) 差不多高. 如果你朋友有这么高和完美的调, 他们马上就会被分心.

下面的 Matlab 代码片段应用了 MDCT 和量化, 接着是如 11.4 节所述的快速去量化和关于折叠部分的逆 MDCT 变换. 使用这样的方式, 我们可以看到伴随有损压缩的量化误差的效应.

```
n=32;                             % length of window
nb=127;                           % number of windows; must be > 1
b=4; L=5;                         % quantization information
q=2*L/(2^b-1);                    % b bits on interval [-L, L]
for i=1:n                         % form the MDCT matrix
  for j=1:2*n
    M(i,j)= cos((i-1+1/2)*(j-1+1/2+n/2)*pi/n);
  end
end
M=sqrt(2/n)*M;
N=M';                             % inverse MDCT
Fs=8192;f=7;                      % Fs=sampling rate
x=cos((1:4096)*pi*64*f/4096);     % test signal
sound(x,Fs)                       % Matlab's sound command
out=[];
for k=1:nb                        % loop over windows
  x0=x(1+(k-1)*n:2*n+(k-1)*n)';
```

```
    y0=M*x0;
    y1=round(y0/q);                  % transform components quantized
    y2=y1*q;                         % and dequantized
    w(:,k)=N*y2;                     % invert the MDCT
    if(k>1)
       w2=w(n+1:2*n,k-1);w3=w(1:n,k);
       out=[out;(w2+w3)/2];          % collect the reconstructed signal
    end
  end
  pause(1)
  sound(out,Fs)                      % play the reconstructed tone
```

上述程序代码播放了一个 1/2 秒 448Hz 的声音, 然后是重构的声音. 比较由程序代码中变量b给出的表示变换部分的位数改变的效果.

建议习题

1. 用奇的 f 与偶的相比, MDCT 变换输出的结果有什么不同? 解释为什么相对偶的 f 而言, 奇的 f 在声音的重构中需要和原始声音差不多的位数.
2. 为程序代码加一个“窗口函数”. 窗口函数在每个窗口的末端把输入的信号光滑地化为 0, 抵消掉问题中信号不完全为周期性的因素. 一个经常的做法是用 x_ih_i 代替 x_i, 其中当窗口长度为 $2n$ 时, 有
$$h_i = \sqrt{2}\sin\frac{(i-1/2)\pi}{2n}$$
为撤回窗口函数的影响, 对 MDCT 逆的输出结果 w_2, w_3 每个部分分别乘上同样的 h_i; 由于窗口函数现在由 1/4 周期来弥补, 这利用到正弦函数的正交性. 比较窗口函数关于为很好地重构声音所需位数的结果.
3. 介绍重要性采样. 构造一个由一些单音组合成的新的测试声音. 修改程序代码使得 y 的每 32 个频率部分有它们自己的量化数 b_k. 给出一种方法, 使得如果平均的 $|y_k|$ 越大, 那么 b_k 越大. 计算保存信号所需的位数, 并且改进你的方案.
4. 写两个独立的子程序: 一个编码, 一个解码. 编码程序要写一个表示量化 MDCT 输出结果的文件 (或 MATLAB 变量) 和输出使用的位数. 解码程序要调入一个由编码程序写的文件并重构信号.
5. 用 Matlab 的命令`wavread`调入一个. `wav`文件, 或者调入其他音频文件 (可以用`handel`. 如果调入的是一个立体音的文件, 那么需要分别在每个频道上处理). 给出和应用一种用于确定如 b_k 所表示的最佳分配位的方法. 使用编码程序压缩音频文件, 并用解码程序进行解码. 比较具有不同压缩量结果的音质.
6. 考虑业界用来更有效压缩声音的更多技巧. 例如在立体音频文件中, 有没有比对频道 s_1 和 s_2 单独处理的更好方法? 为什么对 $(s_1+s_2)/2$ 和 $(s_1-s_2)/2$ 压缩有可能有更好的结果?

软件和进一步阅读

关于数据压缩更好的例子介绍, 可以参看 [6, 11, 10]. 关于图像和声音压缩的通用参考书是 [1, 8]. 参考资料 [9] 是获得 DCT 信息的一个很好资源.Huffman 编码最早的文章是 [4].

我们已经介绍了图像压缩 JPEG 标准的基线 [13]. 完整的标准可以在 [7] 中找到. 最新的压缩标准 JPEG-2000 允许用小波分析代替 DCT[12].

很多声音压缩的协议都基于 MDCT[14, 5]. 更详细的关于单格式的信息可以相应地找到, 如 MP3[3], AAC(高级音频编码, 用于 Apple ITunes 和 QuickTime 视频, 以及 XM 卫星广播) 和开放资源的音频格式 Ogg-Vorbis.

第 12 章　特征值和奇异值

使用万维网, 新用户也很容易访问到大量信息: 非常大, 信息量事实上, 具有强大的搜索引擎的导航是必不可少的. 技术上已经能提供小型化和低成本的传感器, 使得研究人员可以使用大量的数据. 如何用有效的方式来使用这些大量的信息?

探索技术和发现知识的许多方面一般得益于转化成特征值或奇异值问题的处理方法. 解这些高维问题的数值方法产生正规较低维子空间投影. 这恰好是简化复杂数据环境最需要的.

实例检验　12.2.3 节后的实例检验 12, 研究被某一著名的网站搜索提供程序使用的、称为世界上最大的持续进行的特征值计算.

求特征值的计算方法基于幂迭代的基本思想, 这是一类特征空间的不动点迭代. 这种思想的成熟形式, 称为 QR 迭代, 是一种求典型矩阵的所有特征值的标准算法.

奇异值分解揭示了矩阵的基本构造, 并且大量用于统计应用中以寻求数据之间的关系. 本章介绍求方阵的特征值和特征向量的方法, 以及一般矩阵的奇异值和奇异向量的方法.

12.1　幂迭代方法

对于计算特征值, 没有直接方法. 情况类似于求根, 这是由于所有可行的方法都与某种类型的迭代有关. 本节, 我们首先考虑问题是否可能化为求根.

附录 A 给出一种计算 $m \times m$ 矩阵的特征值和特征向量的方法, 这种以求 m 次多项式的根为基础的方法, 对 2×2 矩阵效果良好. 对于较大的矩阵, 这种方法需要第 1 章研究过的那种类型的求根方法.

如果我们回想起第 1 章中的 Wilkinson 多项式, 这种求特征值方法的困难就清楚了. 我们已经知道, 多项式系数的很小改变可以任意大地改变多项式的根. 换言之, 把系数带入到根的输入/输出问题的条件数可能极大. 因为特征多项式的系数计算将受制于机器舍入误差或者更大, 所以用这种方法求特征值对大的误差敏感. 这种困难很大, 作为精确计算特征值的路径, 这种求特征多项式的根的方法以致要被取消.

这种方法精度很差的简单例子来自 Wilkinson 多项式的存在. 如果试图求矩阵

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 20 \end{bmatrix} \tag{12.1}$$

的特征值, 我们将计算特征多项式 $P(x) = (x-1)(x-2)\cdots(x-20)$ 的系数, 并用一种求根方法去求根. 然而, 如第 1 章所示, $P(x)$ 的机器形式的某些根与 $P(x)$ 的真实形式的根 (即 A 的特征值) 相差甚远.

亮点　条件作用

"特征多项式方法" 受到的大误差并不是求根方法的错误. 完全精确的求根方法的遭遇不会更好. 当为了输入求根方法, 而把多项式乘开来确定它的系数时, 一般地, 系数将受制于机器 ε 的阶的误差. 然后将要求求根方法去求稍微错误的多项式的根, 就像我们已看到的, 这可能有灾难性的结果. 对这个问题没有一般的解决办法. 克服这个问题的唯一方法可能是增加代表浮点数的尾数的长度, 这有可能降低机器 ε 的影响. 如果能使机器 ε 比 $1/\text{cond}(P)$ 低, 那么就能对特征值保证精度. 当然, 这其实并不是一种解决办法, 仅仅是"一场不可能胜利的军备竞赛"的升级而已. 如果使用更高精度的计算, 那么我们总是可以把 Wilkinson 多项式推广到更高次以求更高的条件数.

本节介绍的方法基于把矩阵的高次幂乘上一个向量, 它一般将随着幂次的增大而转化成特征向量. 我们以后将改进这种思想, 但是在最复杂的方法中保留它的形迹.

12.1.1 幂迭代

幂迭代的动机是通过乘以一个矩阵来把向量朝主特征向量方向移动.

定义 12.1 设 A 是 $m \times m$ 矩阵, A 的**主特征值**(dominant eigenvalue) 是这样一个特征值 λ, 其大于 A 的所有其他特征值. 如果它存在, 那么与 λ 对应的特征向量称为**主特征向量**(dominant eigenvector).

矩阵

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}$$

有主特征值 4 与特征向量 $[1,1]^T$, 以及较小的特征值 -1 与相应的特征向量 $[-3,2]^T$. 我们观察把矩阵 A 乘以 "随机的" 向量 (譬如说 $[-5,5]^T$) 的结果:

$$\boldsymbol{x}_1 = \boldsymbol{A}\boldsymbol{x}_0 = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} -5 \\ 5 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix},$$

$$\boldsymbol{x}_2 = \boldsymbol{A}^2\boldsymbol{x}_0 = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 10 \\ 0 \end{bmatrix} = \begin{bmatrix} 10 \\ 20 \end{bmatrix},$$

$$\boldsymbol{x}_3 = \boldsymbol{A}^3\boldsymbol{x}_0 = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 10 \\ 20 \end{bmatrix} = \begin{bmatrix} 70 \\ 60 \end{bmatrix},$$

$$\boldsymbol{x}_4 = \boldsymbol{A}^4\boldsymbol{x}_0 = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 70 \\ 60 \end{bmatrix} = \begin{bmatrix} 250 \\ 260 \end{bmatrix} = 260 \begin{bmatrix} \frac{25}{26} \\ 1 \end{bmatrix}.$$

用矩阵 $\boldsymbol{A}$ 重复乘以随机初始向量, 其结果是把这个向量移到非常接近 $\boldsymbol{A}$ 的主特征向量. 这不是巧合, 可以通过把 $\boldsymbol{x}_0$ 表示为特征向量的线性组合来证明这一点:

$$\boldsymbol{x}_0 = 1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} -3 \\ 2 \end{bmatrix}.$$

就此而论重新写出以下计算:

$$\boldsymbol{x}_1 = \boldsymbol{A}\boldsymbol{x}_0 = 4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 2 \begin{bmatrix} -3 \\ 2 \end{bmatrix},$$

$$\boldsymbol{x}_2 = \boldsymbol{A}^2\boldsymbol{x}_0 = 4^2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} -3 \\ 3 \end{bmatrix},$$

$$\boldsymbol{x}_3 = \boldsymbol{A}^3\boldsymbol{x}_0 = 4^3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 2 \begin{bmatrix} -3 \\ 2 \end{bmatrix},$$

$$\boldsymbol{x}_4 = \boldsymbol{A}^4\boldsymbol{x}_0 = 4^4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} -3 \\ 2 \end{bmatrix} = 256 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} -3 \\ 2 \end{bmatrix}.$$

其特点是: 与在绝对值上最大的特征值对应的特征向量在若干步之后将主导这个计算. 本例中, 特征值 4 最大, 所以计算朝着方向为 $[1,1]^{\mathrm{T}}$ 的特征向量移动得越来越近.

为了掌控这些数避免失去控制, 必须在每一步标准化向量. 做到这点的一种方法是: 在每一步之前把目前的向量除以它的长度. 这两种运算, 标准化和用 A 相乘, 构成了幂迭代这种方法.

当这些步骤提供了改进的近似特征向量后, 我们如何求近似特征值? 把问题提得更一般些, 假设矩阵 $\boldsymbol{A}$ 和近似特征向量已经知道. 相应的特征值的最佳估计是什么?

我们将求助于最小二乘. 考虑特征值方程 $\boldsymbol{x}\lambda = \boldsymbol{A}\boldsymbol{x}$, 这里 $\boldsymbol{x}$ 是近似特征向量, 而 λ 未知. 考虑这个方法, 系数矩阵是 $n \times 1$ 矩阵 $\boldsymbol{x}$. 正规方程表明最小二乘的答案是 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}\lambda = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}$ 的解, 或者

$$\lambda = \frac{\boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}}, \tag{12.2}$$

称之为**Rayleigh 商**. 对于给定的近似特征向量, Raylegh 商是最佳近似特征值. 对标准化特征向量应用 Rayleigh 商就把特征值的近似值加到幂迭代.

幂迭代

给定初始向量 $\boldsymbol{x}_0$.

for $j = 1, 2, 3, \cdots$

$\boldsymbol{u}_{j-1} = \boldsymbol{x}_{j-1} / \|\boldsymbol{x}_{j-1}\|$

$\boldsymbol{x}_j = \boldsymbol{A}\boldsymbol{u}_{j-1}$

$\lambda_j = \boldsymbol{u}_{j-1}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{u}_{j-1}$

end

要求矩阵 $\boldsymbol{A}$ 的主特征向量, 从初始向量开始. 每一次迭代包括对目前向量的标准化和用 $\boldsymbol{A}$ 相乘. Rayleigh 商用于近似特征值. MATLAB **norm**命令使它执行简单, 如以下代码所示:

```
% Program 12.1 Power Iteration
% Computes dominant eigenvector of square matrix
% Input: matrix A, initial (nonzero) vector x, number of steps k
% Output: dominant eigenvector x, eigenvalue lam
function [lam,x]=powerit(A,x,k)
for j=1:k
    u=x/norm(x);                   % normalize vector
    x=A*u;                         % power step
    lam=u'*x;                      % Rayleigh quotient
end
```

亮点　收敛性

幂迭代本质上是在每一步进行标准化的不动点迭代. 像 FPI 一样, 它是线性收敛, 就是说当收敛时, 在每一个迭代步误差通过常数因子减小. 在本节后面, 我们将遇到二次收敛的幂迭代的变形, 它称为 Rayleigh 商迭代.

12.1.2 幂迭代的收敛性

我们将在特征值的某些条件下证明幂迭代的收敛性. 尽管这些条件不完全具有一般性, 但是它们可用来证明为什么该方法在这种最明显的情形下是成功的. 以后我们将不断收集更复杂的特征值方法, 它们建立在幂迭代的基本原理上, 可适用于更一般的矩阵.

定理 12.2 设 $\boldsymbol{A}$ 是 $m \times m$ 矩阵, 有实特征值 $\lambda_1, \cdots, \lambda_m$ 满足 $|\lambda_1| > |\lambda_2| \geqslant$

$|\lambda_3| \geqslant \cdots \geqslant |\lambda_m|$. 假设 $\boldsymbol{A}$ 的特征向量张成 $\mathbf{R}^m$. 对几乎每一个初始向量, 幂迭代线性收敛于相应于 λ_1 的特征向量, 其收敛速度常数 $S = |\dfrac{\lambda_2}{\lambda_1}|$.

证 设 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ 是形成 $\mathbf{R}^n$ 的一组基的特征向量, 它们分别相应于特征值 $\lambda_1, \cdots, \lambda_n$. 把初始向量在这组基上表示为 $\boldsymbol{x}_0 = c_1\boldsymbol{v}_1 + \cdots + c_n\boldsymbol{v}_n$, 系数分别是 c_i. "对几乎每一个初始向量" 意思是我们可以假设 $c_1, c_2 \neq 0$. 利用幂迭代得到

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{x}_0 &= c_1\lambda_1\boldsymbol{v}_1 + c_2\lambda_2\boldsymbol{v}_2 + \cdots + c_n\lambda_n\boldsymbol{v}_n,\\
\boldsymbol{A}^2\boldsymbol{x}_0 &= c^1\lambda_1^2\boldsymbol{v}_1 + c_2\lambda_2^2\boldsymbol{v}_2 + \cdots + c_n\lambda_n^2\boldsymbol{v}_n,\\
\boldsymbol{A}^3\boldsymbol{x}_0 &= c^1\lambda_1^3\boldsymbol{v}_1 + c_2\lambda_2^3\boldsymbol{v}_2 + \cdots + c_n\lambda_n^3\boldsymbol{v}_n,\\
&\ \vdots
\end{aligned}$$

在每一步进行标准化. 当步数 $k \to \infty$ 时, 无论怎样进行标准化, 右边的第一项都将占优, 这是因为

$$\frac{\boldsymbol{A}^k\boldsymbol{x}_0}{\lambda_1^k} = c_1\boldsymbol{v}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\boldsymbol{v}_2 + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\boldsymbol{v}_n.$$

对 $i > 1, |\lambda_1| > |\lambda_i|$ 的假设意味着右边除了第一项以外的所有项将以收敛速度 $S \leqslant |\frac{\lambda_2}{\lambda_1}|$ 收敛到零, 而且只要 $c_2 \neq 0$, 恰好就是那个收敛速度 S. 结果, 这个方法以收敛于主特征向量 $\boldsymbol{v}_1$ 的一个倍数, 其特征值是 λ_1.

在定理结论中的术语 "几乎每一个" 意思是使迭代失败的初始向量集合是 $\mathbf{R}^m$ 中较低维的集合. 特别地, 如果 $\boldsymbol{x}_0$ 不包括在由 $[\boldsymbol{v}_1, \boldsymbol{v}_3, \cdots, \boldsymbol{v}_m]$ 及 $[\boldsymbol{v}_2, \boldsymbol{v}_3, \cdots, \boldsymbol{v}_m]$ 张成的 $(m-1)$ 维平面的组合之内, 那么迭代将以确定的速度收敛.

12.1.3 逆幂迭代

幂迭代限于确定 (绝对值) 最大的特征值. 如果把幂迭代用于这个矩阵的逆, 那么就能找到最小的特征值.

引理 12.3 设 $m \times m$ 矩阵 $\boldsymbol{A}$ 的特征值用 $\lambda_1, \lambda_2, \cdots, \lambda_m$ 表示. (a) 逆矩阵 $\boldsymbol{A}^{-1}$ 的特征值是 $\lambda_1^{-1}, \lambda_2^{-2}, \cdots, \lambda_m^{-1}$, 假定逆矩阵存在. 特征向量与 $\boldsymbol{A}$ 的特征向量相同. (b) 移位矩阵 $\boldsymbol{A} - s\boldsymbol{I}$ 的特征值是 $\lambda_1 - s, \lambda_2 - s, \cdots, \lambda_m - s$ 并且特征向量与 $\boldsymbol{A}$ 的特征向量相同.

证 (a) $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$ 意味着 $\boldsymbol{v} = \lambda\boldsymbol{A}^{-1}\boldsymbol{v}$, 因此 $\boldsymbol{A}^{-1}\boldsymbol{v} = \left(\frac{1}{\lambda}\right)\boldsymbol{v}$. 注意特征向量没有改变. (b) 从 $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$ 两边减去 $s\boldsymbol{I}\boldsymbol{v}$. 那么 $(\boldsymbol{A} - s\boldsymbol{I})\boldsymbol{v} = (\lambda - s)\boldsymbol{v}$ 是 $(\boldsymbol{A} - s\boldsymbol{I})$ 的特征值定义, 并且还能用相同的特征向量.

根据引理 12.3, 矩阵 $\boldsymbol{A}^{-1}$ 的绝对值最大的特征值是 $\boldsymbol{A}$ 的绝对值最小的特征值的倒数. 把幂迭代用到逆矩阵. 把结果得到的 $\boldsymbol{A}^{-1}$ 的特征值反过来, 给出 $\boldsymbol{A}$ 的绝对值最小的特征值.

为了避免 $\boldsymbol{A}$ 的逆的显式的计算, 我们对 $\boldsymbol{A}^{-1}$ 重新应用幂迭代, 即

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}^{-1}\boldsymbol{x}_k, \tag{12.3}$$

写成等价的

$$\boldsymbol{A}\boldsymbol{x}_{k+1} = \boldsymbol{x}_k, \tag{12.4}$$

然后可以用 Gauss 消去法解出 x_{k+1}.

现在我们知道如何求矩阵的最大和最小特征值. 换言之, 对一个 100×100 矩阵, 我们完成了 2%, 如何求其余的 98% 呢?

一种方法是受到引理 12.3(b) 的启发. 可以用接近特征值的数值通过对 $\boldsymbol{A}$ 进行移位, 使任何其他的特征值变小. 如果我们碰巧知道有一个特征值靠近 10(譬如说 10.05), 那么 $\boldsymbol{A} - 10\boldsymbol{I}$ 有一个特征值 $\lambda = 0.05$. 如果它是 $\boldsymbol{A} - 10\boldsymbol{I}$ 的绝对值最小的特征值, 那么逆幂迭代 $\boldsymbol{x}_{k+1} = (\boldsymbol{A} - 10\boldsymbol{I})^{-1}\boldsymbol{x}_k$ 将确定它. 也就是说, 逆幂迭代将收敛于倒数 $\frac{1}{0.05} = 20$, 在我们把它倒过来成为 0.05 并且加回移位就得到 10.05. 这个诀窍将确定移位后最小的特征值: 它是表明特征值最接近移位的另一种方法. 总结一下, 我们写

逆幂迭代

给定初始向量 $\boldsymbol{x}_0$ 以及移位 s

for $j = 1, 2, 3, \cdots$

$\boldsymbol{u}_{j-1} = \boldsymbol{x}_{j-1}/\|\boldsymbol{x}_{j-1}\|$

解 $(\boldsymbol{A} - s\boldsymbol{I})\boldsymbol{x}_j = \boldsymbol{u}_{j-1}$

$\lambda_j = \boldsymbol{u}_{j-1}^{\mathrm{T}}\boldsymbol{x}_j$

end

要求 $\boldsymbol{A}$ 的最接近实数 s 的特征值, 对 $(\boldsymbol{A} - s\boldsymbol{I})^{-1}$ 用幂迭代得到 $(\boldsymbol{A} - s\boldsymbol{I})^{-1}$ 的绝对值最大的特征值 $\boldsymbol{b}$. 幂迭代应该通过对 $(\boldsymbol{A} - s\boldsymbol{I})\boldsymbol{y}_{k+1} = \boldsymbol{x}_k$ 用 Gauss 消去法求解而完成. 那么 $\lambda = \boldsymbol{b}^{-1} + s$ 就是 $\boldsymbol{A}$ 的最接近 s 的特征值. 与 λ 对应的特征向量直接从计算中给出.

```
% Program 12.2 Inverse Power Iteration
% Computes eigenvalue of square matrix nearest to input s
% Input: matrix A, (nonzero) vector x, shift s, steps k
% Output: dominant eigenvector of inv (A-sI), eigenvalue lam
function [lam,x]=invpowerit(A,x,s,k)
As=A-s*eye(size(A));
for j=1:k
  u=x/norm(x);               % normalize vector
  x=As\u;                    % power step
  lam=u'*x;                  % Rayleigh Quotient
end
lam=1/lam+s;
```

例 12.1 假设 $\boldsymbol{A}$ 是 5×5 矩阵, 其特征值是 $-5,-2,\frac{1}{2},\frac{3}{2},4$. 当使用 (a) 幂迭代; (b) 逆幂迭代, 移位 $s=0$; (c) 逆幂迭代, 移位 $s=2$ 时, 求特征值和预期的收敛速度.

(a) 带有随机初始向量的幂迭代收敛到绝对值最大的特征值 -5, 收敛速度 $s=\frac{|\lambda_2|}{|\lambda_1|}=\frac{4}{5}$. (b) 逆幂迭代 (没有移位) 将收敛到最小的特征值 $\frac{1}{2}$, 这是因为它的倒数比其他倒数 $-\frac{1}{5},-\frac{1}{2},\frac{2}{3},\frac{1}{4}$ 大. 收敛速度将是逆矩阵的两个最大特征值的比,$s=(\frac{\frac{2}{3}}{2}=\frac{1}{3}$. (c) 带有移位 $s=2$ 的逆幂迭代将确定最接近 2 的特征值, 它是 $\frac{3}{2}$. 其理由是: 特征值移位到 $-7,-4,-\frac{3}{2},-\frac{1}{2},2$ 之后, 倒数的最大者是 -2. 求倒数后得到 $-\frac{1}{2}$, 再加回移位 $s=2$, 我们得到 $\frac{3}{2}$. 收敛速度还是比 $(\frac{2}{3})/2=\frac{1}{3}$. ◀

12.1.4 Rayleigh 商迭代

Rayleigh 商可以与逆幂迭代结合起来使用. 我们知道它收敛到对应于与移位 s 距离最小的特征值的特征值, 而且如果这个距离小, 那么收敛快. 如果循着方法的任一步知道近似特征值, 那么可以把它用作移位 s 来加速收敛.

把 Rayleigh 商用作逆幂迭代中的最新移位导致 Rayleigh 商迭代(RQI).

Rayleight 商迭代

给定初始向量 $\boldsymbol{x}_0$.
for $j=1,2,3,\cdots$
 $\boldsymbol{u}_{j-1}=\boldsymbol{x}_{j-1}/\|\boldsymbol{x}_{j-1}\|$
 $\lambda_{j-1}=\boldsymbol{u}_{j-1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{u}_{j-1}$
 解 $(\boldsymbol{A}-\lambda_{j-1}\boldsymbol{I})\boldsymbol{x}_j=\boldsymbol{u}_{j-1}$
end

```
% Program 12.3 Rayleigh Quotient Iteration
% Input: matrix A, initial (nonzero) vector x, number of steps k
% Output: eigenvalue lam and eigenvector x
function [lam,x]=rqi(A,x,k)
for j=1:k
  u=x/norm(x);                    % normalize
  lam=u'*A*u;                     % Rayleigh quotient
  x=(A-lam*eye(size(A)))\u;       % inverse power iteration
end
x=x/norm(x);
lam=x'*A*x;                       % Rayleigh quotient
```

虽然逆幂迭代线性收敛, 但是 Rayleigh 商迭代对于简单的 (非重) 特征值是二次收敛的, 而且如果矩阵对称它将是三次收敛的. 这就意味着这种方法收敛到机器精度只需要很少步. 收敛之后, 矩阵 $\boldsymbol{A}-\lambda_{j-1}\boldsymbol{I}$ 是奇异的而且不必执行更多的步骤. 注意对 RQI 而言复杂性已经产生. 逆幂迭代仅需要一次 LU 分解; 但是对于 RQI,

因为移位已改变, 故每一步需要一次新的分解. 即使如此, Rayleigh 商迭代是本节中提出的每一次求一个特征值的收敛最快的方法. 在 12.2 节中, 我们讨论在同样的计算中求矩阵的所有特征值的方法. 基本工具将保留幂迭代: 这仅是会变得更复杂的组织细节.

习题 12.1

1. 求以下对称矩阵的特征多项式、特征值以及特征向量:

(a) $\begin{bmatrix} 3.5 & -1.5 \\ -1.5 & 3.5 \end{bmatrix}$; **(b)** $\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$; **(c)** $\begin{bmatrix} -0.2 & -2.4 \\ -2.4 & 1.2 \end{bmatrix}$; **(d)** $\begin{bmatrix} 136 & -48 \\ -48 & 164 \end{bmatrix}$.

2. 求以下矩阵的特征多项式、特征值以及特征向量:

(a) $\begin{bmatrix} 7 & 9 \\ -6 & -8 \end{bmatrix}$; **(b)** $\begin{bmatrix} 2 & 6 \\ 1 & 3 \end{bmatrix}$; **(c)** $\begin{bmatrix} 2.2 & 0.6 \\ -0.4 & 0.8 \end{bmatrix}$; **(d)** $\begin{bmatrix} 32 & 45 \\ -18 & -25 \end{bmatrix}$.

3. 求以下矩阵的特征多项式、特征值以及特征向量:

(a) $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & -2 \\ 0 & 0 & 2 \end{bmatrix}$; **(b)** $\begin{bmatrix} 1 & 0 & -\frac{1}{3} \\ 0 & 1 & \frac{2}{3} \\ -1 & 1 & 1 \end{bmatrix}$; **(c)** $\begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{6} \\ -1 & 0 & \frac{1}{3} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$.

4. 证明方阵和它的转置有相同的特征多项式, 因此有相同的特征值.

5. 假设 $\boldsymbol{A}$ 是具有给定特征值的 3×3 矩阵. 确定对哪一个特征值而言, 幂迭代将收敛, 并且确定收敛速度常数 S.

(a) $\{3,1,4\}$; **(b)** $\{3,1,-4\}$; **(c)** $\{-1,2,4\}$; **(d)** $\{1,9,10\}$.

6. 假设 $\boldsymbol{A}$ 是给定特征值的 3×3 矩阵. 确定对哪一个特征值, 幂迭代将收敛, 并且确定收敛速度常数 S.

(a) $\{1,2,7\}$; **(b)** $\{1,1,-4\}$; **(c)** $\{0,-2,5\}$; **(d)** $\{8,-9,10\}$.

7. 假设 $\boldsymbol{A}$ 是给定特征值的 3×3 矩阵. 确定对哪个特征值, 带有给定移位 s 的逆幂迭代将收敛, 并且确定收敛速度常数 S.

(a) $\{3,1,4\}, s=0$; **(b)** $\{3,1,-4\}, s=0$;
(c) $\{-1,2,4\}, s=0$; **(d)** $\{1,9,10\}, s=6$.

8. 假设 $\boldsymbol{A}$ 是给定特征值的 3×3 矩阵. 确定对哪一个特征值, 带有给定移位的逆幂迭代将收敛, 并且确定收敛速度常数 S.

(a) $\{3,1,4\}, s=5$; **(b)** $\{3,1,-4\}, s=4$;
(c) $\{-1,2,4\}, s=1$; **(d)** $\{1,9,10\}, s=8$.

9. 设 $\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix}$. (a) 求 $\boldsymbol{A}$ 的所有特征值及特征向量. (b) 取初始向量 $\boldsymbol{x}_0 = [1,0]$, 采用 3 步幂迭代, 在每一步, 用目前的 Rayleigh 商近似特征值. (c) 预测用带有移位 $s=0$ 的逆幂迭代的结果. (d) 用移位 $s=3$, 预测逆幂迭代的结果.

10. 设 $\boldsymbol{A} = \begin{bmatrix} -2 & 1 \\ 3 & 0 \end{bmatrix}$, 对这个矩阵执行习题 9 的步骤.

11. 如果 A 是特征值为 $-6, -3, 1, 2, 5, 7$ 的 6×6 矩阵, 以下算法将求得 A 的哪一个特征值? (a) 幂迭代; (b) 带有移位 $s = 4$ 的逆幂迭代;(c) 求这两种计算的线性收敛速度, 哪一种收敛得较快?

计算机问题 12.1

1. 对幂迭代方法使用提供的代码 (或你自己的代码), 求 A 的主特征向量, 并且通过计算 Rayleigh 商估计主特征值. 把你的结果与习题 5 的相应部分进行比较:

$$\text{(a)} \begin{bmatrix} 10 & -12 & -6 \\ 5 & -5 & -4 \\ -1 & 0 & 3 \end{bmatrix}; \quad \text{(b)} \begin{bmatrix} -14 & 20 & 10 \\ -19 & 27 & 12 \\ 23 & -32 & -13 \end{bmatrix};$$

$$\text{(c)} \begin{bmatrix} 8 & -8 & -4 \\ 12 & -15 & -7 \\ -18 & 26 & 12 \end{bmatrix}; \quad \text{(d)} \begin{bmatrix} 12 & -4 & -2 \\ 19 & -19 & -10 \\ -35 & 52 & 27 \end{bmatrix}.$$

2. 对逆幂迭代方法使用提供的代码 (或你自己的代码), 检验你从习题 7 得出的结果, 可用计算机问题 1 中相应的矩阵.

3. 对逆幂迭代方法, 用计算机问题 1 中相应的矩阵, 检验你从习题 8 得出的结果.

4. 对计算机问题 1 中的矩阵应用 Rayleigh 商迭代, 试用不同的初始向量直到求出全部 3 个特征值.

12.2 QR 算法

本节的目标是建立同时求出全部特征值的方法. 我们从应用于对称矩阵的方法开始, 以后把它补充到一般情况下使用. 因为对称矩阵的特征值是实数并且它们的单位特征向量形成 $\mathbf{R}^m$ 的一组标准正交基, 所以对称矩阵比较容易处理 (见附录 A). 它促使平行地对 m 个向量用幂迭代, 这里我们积极地做到保持向量彼此正交.

12.2.1 同时迭代

假设我们从 m 个两两正交的初始向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$ 开始. 对每一个向量用一步幂迭代后, $\boldsymbol{A}\boldsymbol{v}_1, \cdots, \boldsymbol{A}\boldsymbol{v}_m$ 不再保证彼此正交. 事实上, 进一步用 $\boldsymbol{A}$ 相乘之后, 根据定理 12.2, 它们都可能倾向于收敛到主特征值.

为了避免这一点, 我们在每一步对这 m 个向量重新正交化. m 个向量同时用 $\boldsymbol{A}$ 相乘, 高效地写成矩阵乘积

$$\boldsymbol{A}[\boldsymbol{v}_1|\cdots|\boldsymbol{v}_m].$$

在第 4 章我们发现, 可以把正交化步看成把结果得到的乘积分解为 QR. 如果把基本基向量用作初始向量, 那么幂迭代的第一步通过重新正交化得到的是 $\boldsymbol{A}\boldsymbol{I} = \bar{\boldsymbol{Q}}_1\boldsymbol{R}_1$,

或者

$$\left[A\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \middle| A\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \middle| \cdots \middle| A\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right] = [\bar{q}_1^1|\cdots|\bar{q}_m^1] \begin{bmatrix} r_{11}^1 & r_{12}^1 & \cdots & r_{1m}^1 \\ & r_{22}^1 & & \vdots \\ & & \ddots & \vdots \\ & & & r_{mm}^1 \end{bmatrix}. \tag{12.5}$$

对 $i=1,\cdots,m, \bar{q}_i^1$ 是在幂迭代过程中单位向量的新的正交集. 下面重复这一步:

$$\begin{aligned} A\bar{Q}_1 &= [A\bar{q}_1^1|A\bar{q}_2^1|\cdots|A\bar{q}_m^1] \\ &= [\bar{q}_1^2|\bar{q}_2^2|\cdots|\bar{q}_m^2] \begin{bmatrix} r_{11}^2 & r_{12}^2 & \cdots & r_{1m}^2 \\ & r_{22}^2 & & \vdots \\ & & \ddots & \vdots \\ & & & r_{mm}^2 \end{bmatrix} = \bar{Q}_2 R_2. \end{aligned} \tag{12.6}$$

换言之, 我们已经建立了同时寻求对称矩阵的所有 m 个特征向量的幂迭代的矩阵形式.

标准化的同时迭代

置 $\bar{Q}_0 = I$
for $j = 1, 2, 3, \cdots$
　　$A\bar{Q}_j = \bar{Q}_{j+1}R_{j+1}$
end

在第 j 步, $\bar{Q}_j$ 的各列近似于 A 的特征向量, 而且对角元素 $r_{11}^j, \cdots, r_{mm}^j$ 近似于特征值. 在 MATLAB代码中, 这个我们将称为标准化的同时迭代 (NSI) 的算法, 可以写得非常简洁.

```
% Program 12.4 Normalized Simultaneous Iteration
% Computes eigenvalues/vectors of symmetric matrix
% Input: matrix A, number of steps k
% Output: eigenvalues lam and eigenvector matrix Q
function [lam,Q]=nsi(A,k)
[m,n]=size(A);
Q=eye(m,m);
for j=1:k
    [Q,R]=qr(A*Q);                % QR factorization
end
lam=diag(Q'*A*Q);                 % Rayleigh quotient
```

可以得到更简洁的方法执行标准化的同时迭代. 设 $\bar{\boldsymbol{Q}}_0=\boldsymbol{I}$, 那么 NSI 如下:

$$\begin{aligned}\boldsymbol{A}\bar{\boldsymbol{Q}}_0&=\bar{\boldsymbol{Q}}_1\boldsymbol{R}_1,\\ \boldsymbol{A}\bar{\boldsymbol{Q}}_1&=\bar{\boldsymbol{Q}}_2\boldsymbol{R}_2,\\ \boldsymbol{A}\bar{\boldsymbol{Q}}_2&=\bar{\boldsymbol{Q}}_3\boldsymbol{R}_3,\\ &\vdots\end{aligned}\tag{12.7}$$

考虑相似迭代 $\boldsymbol{Q}_i=\boldsymbol{I}$, 以及

$$\begin{aligned}\boldsymbol{A}_0\equiv\boldsymbol{A}\boldsymbol{Q}_0&=\boldsymbol{Q}_1\boldsymbol{R}_1',\\ \boldsymbol{A}_1\equiv\boldsymbol{R}_0'\boldsymbol{Q}_1&=\boldsymbol{Q}_2\boldsymbol{R}_2',\\ \boldsymbol{A}_2\equiv\boldsymbol{R}_2'\boldsymbol{Q}_2&=\boldsymbol{Q}_3\boldsymbol{R}_3',\\ &\vdots\end{aligned}\tag{12.8}$$

我们将称之为**非移位 QR 算法**. 仅有的不同是在第一步后不再需要 $\boldsymbol{A}$; 它被目前的 $\boldsymbol{R}_k$ 所代替. 比较 (12.7) 和 (12.8) 表明我们可以在 (12.7) 中选取 $\boldsymbol{Q}_1=\bar{\boldsymbol{Q}}_1$ 以及 $\boldsymbol{R}_1=\boldsymbol{R}_1'$. 进而, 由于

$$\bar{\boldsymbol{Q}}_2\boldsymbol{R}_2=\boldsymbol{A}\bar{\boldsymbol{Q}}_1=\boldsymbol{Q}_1\boldsymbol{R}_1'\bar{\boldsymbol{Q}}_1=\boldsymbol{Q}_1\boldsymbol{R}_1'\boldsymbol{Q}_1=\boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{R}_2',\tag{12.9}$$

我们可以在 (12.7) 中选取 $\bar{\boldsymbol{Q}}_2=\boldsymbol{Q}_1\boldsymbol{Q}_2$ 以及 $\boldsymbol{R}_2=\boldsymbol{R}_2^1$. 事实上, 如果我们已经选取 $\bar{\boldsymbol{Q}}_{k-1}=\boldsymbol{Q}_1\cdots\ \boldsymbol{Q}_{k-1}$ 以及 $\boldsymbol{R}_{j-1}=\boldsymbol{R}_{j-1}^1$, 那么

$$\begin{aligned}\bar{\boldsymbol{Q}}_j\boldsymbol{R}_j&=\boldsymbol{A}\bar{\boldsymbol{Q}}_{j-1}=\boldsymbol{A}\boldsymbol{Q}_1\cdots\boldsymbol{Q}_{j-1}=\bar{\boldsymbol{Q}}_2\boldsymbol{R}_2\boldsymbol{Q}_2\cdots\boldsymbol{Q}_{j-1}\\ &=\bar{\boldsymbol{Q}}_2\boldsymbol{Q}_3\boldsymbol{R}_3\boldsymbol{Q}_3\cdots\boldsymbol{Q}_{j-1}=\boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{Q}_3\boldsymbol{Q}_4\boldsymbol{R}_4\boldsymbol{Q}_4\cdots\boldsymbol{Q}_{j-1}\\ &=\cdots=\boldsymbol{Q}_1\cdots\boldsymbol{Q}_j\boldsymbol{R}_j,\end{aligned}\tag{12.10}$$

并且, 我们在 (12.7) 中可以定义 $\bar{\boldsymbol{Q}}_j=\boldsymbol{Q}_1\cdots\boldsymbol{Q}_j$ 以及 $\boldsymbol{R}_j=\boldsymbol{R}_j'$. 因此, 非移位算法用稍微不同的记号与标准化同时迭代做相同的计算. 还注意到

$$\boldsymbol{A}_{j-1}=\boldsymbol{Q}_j\boldsymbol{R}_j=\boldsymbol{Q}_j\boldsymbol{R}_j\boldsymbol{Q}_j\boldsymbol{Q}_j^{\mathrm{T}}=\boldsymbol{Q}_j\boldsymbol{A}_j\boldsymbol{Q}_j^{\mathrm{T}},\tag{12.11}$$

所以所有的 $\boldsymbol{A}_j$ 都是相似矩阵, 因此有相同的特征值.

```
% Program 12.5 Unshifted QR Algorithm
% Computes eigenvalues/vectors of symmetric matrix
% Input: matrix A, number of steps k
% Output: eigenvalues lam and eigenvector matrix Qbar
function [lam,Qbar]=unshiftedqr(A,k)
[m,n]=size(A);
Q=eye(m,m);
Qbar=Q; R=A;
for j=1:k
    [Q,R]=qr(R*Q);             % QR factorization
    Qbar=Qbar*Q;               % accumulate Q's
end
lam=diag(R*Q);                 % diagonal converges to eigenvalues
```

定理 12.4 假设 $\boldsymbol{A}$ 是对称的 $m \times m$ 矩阵，其特征值 λ_i 满足 $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_m|$，非移位 QR 算法线性收敛于 $\boldsymbol{A}$ 的特征向量和特征值. 当 $j \to \infty$ 时，$\boldsymbol{A}_j$ 收敛于特征值在主对角线上的对角矩阵，并且 $\bar{\boldsymbol{Q}}_j = \boldsymbol{Q}_1 \cdots \boldsymbol{Q}_j$ 收敛于一个正交矩阵，它的列是特征向量.

定理 12.4 的证明能在 Golub & Van Loan[5] 中找到. 标准化同时迭代，本质上相同的算法，在相同条件下收敛. 注意，如果定理的条件不满足，那么即使对于对称矩阵非移位 QR 算法也可能失败. 见习题 5.

尽管非移位 QR 算法是幂迭代的改进形式，由定理 12.4 要求的条件是严格的，因此我们需要加一些改进使这个求特征值的方法更通用——例如在非对称矩阵的情形. 同样也出现在对称矩阵中的一个问题是：在主特征向量相持的情形下不能保证非移动 QR 算法起作用. 这种情形的一个例子是

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

它有特征值 1 和 -1. 当特征值是复数时，另一种形式的 "相持" 出现了. 非对称矩阵

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

的特征值是 i 和 $-$i，这两个复数的模都是 1. 非移位 QR 算法的定义中没有允许计算复特征值. 更有甚者，非移位 QR 算法没有使用逆幂迭代的策略. 我们发现用这种策略能够大大加速幂迭代，并且我们想找一种方法把这种思想用到新的执行程序中去. 在介绍了 QR 算法的目标把矩阵 $\boldsymbol{A}$ 简化成它的实 Schur 形式，之后，下面应用这些经过改进的设计.

12.2.2 实 Schur 形式和 QR 算法

QR 算法求矩阵 $\boldsymbol{A}$ 的特征值的方法是确定特征值显然的相似矩阵. 下面的例子是实 Schur 形式.

定义 12.5 *如果矩阵 $\boldsymbol{T}$ 是上三角矩阵, (主对角线上有 2×2 块的除外,) 那么它有***实 Schur 形式**.

例如, 形式为

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & \times & \times & \times \\ & & & & \times \end{bmatrix}$$

的矩阵有实 Schur 形式. 根据习题 6, 这种形式的矩阵的特征值是对角分块矩阵的特征值: 当分块是 1×1 时, 就是对角元素, 或者在 2×2 分块的情形下就是这个 2×2 分块的特征值. 不管哪一种, 矩阵的特征值计算很快.

这个定义的价值是每个实方阵相似于一个这种形式的矩阵. 这是定理 12.6 的结论, 证明在 [5] 中.

定理 12.6 *设 $\boldsymbol{A}$ 是实方阵. 那么存在正交矩阵 $\boldsymbol{Q}$ 以及实 Schur 形式的矩阵 $\boldsymbol{T}$ 使得*

$$\boldsymbol{A} = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{T}\boldsymbol{Q}.$$

所谓矩阵 $\boldsymbol{A}$ 的 Schur 分解是一种 "特征值–展现分解", 意思是如果能够执行它, 我们将知道特征值和特征向量.

完整的 QR 算法通过一系列相似变换, 反复地把任意矩阵 $\boldsymbol{A}$ 朝它的 Schur 分解移动. 我们将分两步进行. 首先, 我们将建立带有移位的逆幂迭代的思想并且加入压缩的概念来建立移位 QR 算法. 然后, 我们将建立允许复特征值的改进形式.

移位形式是直接写的. 每一步包括应用移位、完成 QR 分解以及再取移位反向. 用符号表示

$$\begin{aligned} \boldsymbol{A}_0 - s\boldsymbol{I} &= \boldsymbol{Q}_1\boldsymbol{R}_1, \\ \boldsymbol{A}_1 &= \boldsymbol{R}_1\boldsymbol{Q}_1 + s\boldsymbol{I}, \end{aligned} \tag{12.12}$$

注意

$$\begin{aligned} \boldsymbol{A}_1 - s\boldsymbol{I} &= \boldsymbol{R}_1\boldsymbol{Q}_1 \\ &= \boldsymbol{Q}_1^{\mathrm{T}}(\boldsymbol{A}_o - s\boldsymbol{I})\boldsymbol{Q}_1 \end{aligned}$$

$$= Q_1^{\mathrm{T}} A_0 Q_1 - sI$$

意味着 A_1 相似于 A_0, 所以有相同的特征值. 重复这一步, 产生矩阵序列 A_k, 它们都相似于 $A = A_0$.

对移位 s, 好的选择是什么? 这把我们引向关于特征值计算的压缩的概念. 我们将选择移位是矩阵 A_k 的底行右边的元素. 这将使迭代 (当它收敛于实 Schur 形式) 把底行除了最右边的元素外转移成一行零. 在这个元素已经收敛到特征值后, 我们通过消去最后的行与列对矩阵进行压缩. 然后我们着手求其余的特征值.

移位 QR 算法的最初尝试在程序 12.6 中所示的 MATLAB代码中给出. 在每一步, 我们用一步移位 QR, 然后检查底行. 如果除了对角元素 a_{nn} 之外的其他元素都小, 那么我们断言这个元素就是特征值, 且对于以后的计算通过略去最后一行和最后一列进行压缩.

在定理 12.4 的假设下, 这个程序将成功. 度量相等的复特征值或者实特征值将造成问题, 这一点我们将在后面更复杂的形式中解决. 习题 7 说明了这种初步形式的 QR 算法的缺点.

```
% Program 12.6 Shifted QR Algorithm, preliminary version
% Computes eigenvalues of matrices without equal size eigenvalues
% Input: matrix a
% Output: eigenvalues lam
function lam=shiftedqr0(a)
tol=1e-14;
m=size(a,1);lam=zeros(m,1);
n=m;
while n>1
    while max(abs(a(n,1:n-1)))>tol
        mu=a(n,n);       % define shift mu
        [q,r]=qr(a-mu*eye(n));
        a=r*q+mu*eye(n);
    end
    lam(n)=a(n,n);       % declare eigenvalue
    n=n-1;               % decrement n
    a=a(1:n,1:n);        % deflate
end
lam(1)=a(1,1);           % 1x1 matrix remains
```

最后, 为了考虑到复特征值的计算, 我们必须允许实 Schur 形式的对角线上存在 2×2 分块. 程序 12.7 中给出的改进形式的移位 QR 算法, 试图把矩阵迭代为右下角是 1×1 的对角分块; 如果它失败了 (在使用者确定的尝试次数之后), 那么它断言, 为一个 2×2 分块, 求出这一组特征值, 然后用 2 压缩. 这种改进形式对绝大多数输入矩阵将收敛于实 Schur 形式, 但并不是全部. 为了把最后少数几个“坚持者”集合起来以及使算法更有效, 我们在 12.2.3 节建立上 Hessenberg 形式.

```
% Program 12.7 Shifted QR Algorithm, general version
% Computes real and complex eigenvalues of square matrix
% Input: matrix a
% Output: eigenvalues lam
function lam=shiftedqr(a)
tol=1e-14;kounttol=500;
m=size(a,1);lam=zeros(m,1);
n=m;
while n>1
    kount=0;
    while max(abs(a(n,1:n-1)))>tol & kount<kounttol
        kount=kount+1;      % keep track of number of qr's
        mu=a(n,n);          % shift is mu
        [q,r]=qr(a-mu*eye(n));
        a=r*q+mu*eye(n);
    end
    if kount<kounttol           % have isolated 1x1 block
        lam(n)=a(n,n);          % declare eigenvalue
        n=n-1;
        a=a(1:n,1:n);           % deflate by 1
    else                        % have isolated 2x2 block
        disc=(a(n-1,n-1)-a(n,n))^2+4*a(n,n-1)*a(n-1,n);
        lam(n)=(a(n-1,n-1)+a(n,n)+sqrt(disc))/2;
        lam(n-1)=(a(n-1,n-1)+a(n,n)-sqrt(disc))/2;
        n=n-2;
        a=a(1:n,1:n);           % deflate by 2
    end
end
if n>0;lam(1)=a(1,1);end      % only a 1x1 block remains
```

即使用它的一般形式, 移位 QR 算法对下面的例子失败:

$$\boldsymbol{A}=\begin{bmatrix}0&0&0&1\\0&0&-1&0\\0&1&0&0\\-1&0&0&0\end{bmatrix}\tag{12.13}$$

像这种带有有重的复特征值的矩阵, 通过移位 QR 可能不向实 Schur 形式转移. 对这些更困难的例子, 所需要的额外的帮助是用相似的上 Hessenberg 形式的矩阵代替 $\boldsymbol{A}$. 这是 12.2.3 节的中心.

12.2.3 上 Hessenberg 形式

如果我们首先把 $\boldsymbol{A}$ 化为上 Hessenberg 形式, 那么 QR 算法的效率可以得到极大的提高.

在开始 QR 迭代之前, 这个思想是, 使用相似变换, 在保持全部特征值的同时, 把尽可能多的零放入 $\boldsymbol{A}$ 中. 此外, 上 Hessenberg 形式将消除我们已建立的这种形

式的 QR 算法的最终困难, 通过保证 QR 迭代将永远转到 1×1 或者 2×2 分块, 从而收敛到复特征值.

定义 12.7 *若 $a_{ij}=0, i>j+1$, 那么 $m\times n$ 矩阵 $\boldsymbol{A}$ 是上 Hessenberg 形式.*

形式为

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

的矩阵是上 Hessenberg 形式. 通过相似交换, 总有某种有限算法把矩阵转化为上 Hessenberg 形式.

定理 12.8 *设 $\boldsymbol{A}$ 是方阵. 存在正交矩阵 $\boldsymbol{Q}$, 使得 $\boldsymbol{A}=\boldsymbol{Q}\boldsymbol{B}\boldsymbol{Q}^{\mathrm{T}}$, 而且 $\boldsymbol{B}$ 是上 Hessenberg 形式.*

通过使用 4.3 节的 Housholder 反射 (在那里它们用于构造 QR 分解), 我们将构造 $\boldsymbol{B}$. 但是, 存在一个主要区别: 现在我们关心用反射 $\boldsymbol{H}$ 乘在矩阵的左边和右边, 因为我们想要以具有相同特征值的相似矩阵结束. 因此, 我们必须对我们可以安装到 $\boldsymbol{A}$ 中的零不太积极.

定义 $\boldsymbol{x}$ 是由 $\boldsymbol{A}$ 的除了第一个元素以外的第一列组成的 $n-1$ 维向量. 设 $\hat{\boldsymbol{H}}_1$ 是把 $\boldsymbol{x}$ 转移为 $(\pm\|\boldsymbol{x}\|, 0, \cdots, 0)$ 的 Hourseholder 反射 (如第 4 章所提到的, 我们应选符号为 -sign$(\boldsymbol{x}_1)$ 以避免在实践中问题相消, 但是这个理论对任一种选择都成立). 设 $\boldsymbol{H}_1$ 是通过把 $\hat{\boldsymbol{H}}_1$ 插入到 $n\times n$ 单位矩阵的底 $(n-1)\times(n-1)$ 角形成的正交矩阵, 那么我们有

$$\boldsymbol{H}_1\boldsymbol{A}=\left[\begin{array}{c|cccc} 1 & 0 & 0 & 0 & 0 \\ \hline 0 & & & & \\ 0 & & \hat{H}_1 & & \\ 0 & & & & \\ 0 & & & & \end{array}\right]\left[\begin{array}{c|cccc} \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array}\right]=\left[\begin{array}{c|cccc} \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array}\right].$$

在我们可以对成功地把零放入矩阵进行求值前, 需要通过在右边乘以 $\boldsymbol{H}_1^{-1}$ 而完成相似变换. 回想起 Householder 反射是对称正交矩阵, 所以 $\boldsymbol{H}_1^{-1}=\boldsymbol{H}_1^{\mathrm{T}}=\boldsymbol{H}_1$. 因

此,

$$H_1AH_1=\left[\begin{array}{c|cccc}\times&\times&\times&\times&\times\\\hline\times&\times&\times&\times&\times\\0&\times&\times&\times&\times\\0&\times&\times&\times&\times\\0&\times&\times&\times&\times\end{array}\right]\left[\begin{array}{c|cccc}1&0&0&0&0\\\hline 0&&&&\\0&&&&\\0&&\hat{H}_1&&\\0&&&&\end{array}\right]=\left[\begin{array}{c|cccc}\times&\times&\times&\times&\times\\\hline\times&\times&\times&\times&\times\\0&\times&\times&\times&\times\\0&\times&\times&\times&\times\\0&\times&\times&\times&\times\end{array}\right].$$

在 H_1A 中产生的零在 H_1AH_1 中并没有改变. 然而, 注意如果我们已试图消掉第一列中一个非零元素以外的所有元素, 就像在 12.2 节的 QR 分解中所做的那样, 那么在右边相乘时, 我们可能不能保持这些零. 事实上, 不存在计算任意矩阵和上三角矩阵之间的相似变换的有限算法. 如果有的话, 本章可能要短得多, 这是因为我们可以从这相似的上三角矩阵的对角线上读出这个任意矩阵的特征值.

实现上 Hessenberg 形式的下一步是重复前一步, 利用由第二列中较底下的 $n-2$ 个元素组成的 $(n-2)$ 维向量 $\boldsymbol{x}$. 令 $\hat{\boldsymbol{H}}_2$ 是针对新的 $\boldsymbol{x}$ 的 $(n-2)\times(n-2)$Householder 反射, 并定义 $\boldsymbol{H}_2$ 是用 $\hat{\boldsymbol{H}}_2$ 代入底角的单位矩阵. 那么

$$H_2(H_1AH_1)=\left[\begin{array}{cc|ccc}1&0&0&0&0\\0&1&0&0&0\\\hline 0&0&&&\\0&0&&\hat{H}_2&\\0&0&&&\end{array}\right]\left[\begin{array}{cc|ccc}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\hline 0&\times&\times&\times&\times\\0&\times&\times&\times&\times\\0&\times&\times&\times&\times\end{array}\right]=\left[\begin{array}{cc|ccc}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\hline 0&\times&\times&\times&\times\\0&0&\times&\times&\times\\0&0&\times&\times&\times\end{array}\right],$$

接着检查上式, 像 H_1 一样, 在右边乘上 $\boldsymbol{H}_2$ 并不反过来影响已得到的零. 如果 $n=5$, 那么再进行一步, 我们得到 5×5 矩阵

$$H_3H_2H_1AH_1^{\mathrm{T}}H_2^{\mathrm{T}}H_3^{\mathrm{T}}=H_3H_2H_1A(H_3H_2H_1)^{\mathrm{T}}=QAQ^{\mathrm{T}},$$

它是上 Hessenberg 形式. 因为这个矩阵相似于 $\boldsymbol{A}$, 所以它有与 $\boldsymbol{A}$ 相同的特征值和重数. 一般地, 对 $n\times n$ 矩阵的 $\boldsymbol{A}$, 需要 $n-2$ 个 Householder 步把 $\boldsymbol{A}$ 转化成上 Hessenberg 形式.

例 12.2 把 $\begin{bmatrix}2&1&0\\3&5&-5\\4&0&0\end{bmatrix}$ 转化成上 Hessenberg 形式.

设 $\boldsymbol{x}=[3,4]$. 前面我们求得 Houreholder 反射

$$\hat{H}_1x=\begin{bmatrix}0.6&0.8\\0.8&-0.6\end{bmatrix}\begin{bmatrix}3\\4\end{bmatrix}=\begin{bmatrix}5\\0\end{bmatrix}.$$

因此,

$$H_1A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.6 & 0.8 \\ 0 & 0.8 & -0.6 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & 5 & -5 \\ 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 \\ 5 & 3 & -3 \\ 0 & 4 & -4 \end{bmatrix},$$

$$A' \equiv H_1AH_1 = \begin{bmatrix} 2 & 1 & 0 \\ 5 & 3 & -3 \\ 0 & 4 & -4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.6 & 0.8 \\ 0 & 0.8 & -0.6 \end{bmatrix} = \begin{bmatrix} 2.0 & 0.6 & 0.8 \\ 5.0 & -0.6 & 4.2 \\ 0.0 & -0.8 & 5.6 \end{bmatrix}.$$

结果是上 Hessenberg 形式的矩阵 A' 并且相似于 A. ◀

下面利用 Householder 反射, 执行前面的策略并且构造一种求 Q 的算法:

```
% Program 12.8 Upper Hessenberg form
% Input: matrix a
% Output: Hessenberg form matrix a and reflectors v
% Usage: [a,v]=hessen(a) yields similar matrix a of
%   Hessenberg form and a matrix v whose columns hold
%   the v's defining the Householder reflectors.
function [a,v]=hessen(a)
[m,n]=size(a);
v=zeros(m,m);
for k=1:m-2
  x=a(k+1:m,k);
  v(1:m-k,k)=-sign(x(1)+eps)*norm(x)*eye(m-k,1)-x;
  v(1:m-k,k)=v(1:m-k,k)/norm(v(1:m-k,k));
  a(k+1:m,k:m)=a(k+1:m,k:m)-2*v(1:m-k,k)*v(1:m-k,k)'*a(k+1:m,k:m);
  a(1:m,k+1:m)=a(1:m,k+1:m)-2*a(:,k+1:m)*v(1:m-k,k)*v(1:m-k,k)';
end
```

对于特征值计算, 上 Hessenberg 形式的一个优点是: 在 QR 算法中, 沿着对角线仅仅出现 2×2 分块, 消除了 12.2.2 节的有重的复特征值造成的困难.

例 12.3 求矩阵 (12.13) 的特征值.

对于

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix},$$

由 Householder 反射给出的上 Hessenberg 形式的相似矩阵是

$$A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

这里 $\boldsymbol{A}' = \boldsymbol{Q}\boldsymbol{A}\boldsymbol{Q}^{\mathrm{T}}$,

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

矩阵 $\boldsymbol{A}'$ 已经是实 Schur 形式. 它的特征值是沿着主对角线上的两个 2×2 矩阵的特征值, 它们是有重的 $\{\mathrm{i}, -\mathrm{i}\}$ 组. ◀

因此, 我们最后有了求任意方阵 $\boldsymbol{A}$ 的全部特征值的完整方法. 首先用相似变换 (程序 12.8) 把矩阵转化成上 Hessenberg 形式, 然后再用移位 QR 算法 (程序 12.7). MATLAB的`eig` 命令提供了基于这一系列计算的精确的特征值.

还有许多其他技巧来加速在这里没有覆盖的 QR 算法的收敛. QR 算法是为满矩阵设计的. 对于大型稀疏系统, 一些替代方法通常将更有效, 见 [9].

习题 12.2

1. 把以下矩阵转化成上 Hessenberg 形式:

(a) $\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$; (c) $\begin{bmatrix} 2 & 1 & 0 \\ 4 & 1 & 1 \\ 3 & 0 & 1 \end{bmatrix}$; (d) $\begin{bmatrix} 1 & 1 & 0 \\ 2 & 3 & 1 \\ 2 & 1 & 0 \end{bmatrix}$.

2. 把矩阵 $\begin{bmatrix} 1 & 0 & 2 & 3 \\ -1 & 0 & 5 & 2 \\ 2 & -2 & 0 & 0 \\ 2 & -1 & 2 & 0 \end{bmatrix}$ 转化成上 Hessenberg 形式.

3. 证明 Hessenberg 形式的对称矩阵是三对角的.

4. 如果一个非负方阵的每列元素加起来等于 1, 就称它是**随机的**. 证明随机矩阵 (a) 有一个特征值等于 1, 而且 (b) 所有的特征值的绝对值最多是 1.

5. 对下列矩阵执行标准化同时迭代, 并解释它如何失败:

(a) $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$.

6. (a) 证明实 Schur 形式的矩阵的行列式是在主对角线上的 1×1 和 2×2 分块矩阵的行列式的乘积. (b) 证明实 Schur 形式的矩阵的特征值是在主对角线上的 1×1 和 $2\times$ 分块矩阵的特征值.

7. 确定初步形式的 QR 算法不论在改变成 Hessenberg 形式之前还是之后, 是否能求出准确的特征值.

(a) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.

8. 确定一般形式的 QR 算法, 不论在改变成 Hessenberg 形式之前还是之后, 对习题 7 中的矩阵是否能求出准确的特征值.

计算机问题 12.2

1. 对以下矩阵直接应用移位 QR 算法 (初步形式Shiftedqr), 误差容限是 10^{-14}:

(a) $\begin{bmatrix} -3 & 3 & 5 \\ 1 & -5 & -5 \\ 6 & 6 & 4 \end{bmatrix}$; (b) $\begin{bmatrix} 3 & 1 & 2 \\ 1 & 3 & -2 \\ 2 & 2 & 6 \end{bmatrix}$; (c) $\begin{bmatrix} 17 & 1 & 2 \\ 1 & 17 & -2 \\ 2 & 2 & 20 \end{bmatrix}$;

(d) $\begin{bmatrix} -7 & -8 & 1 \\ 17 & 18 & -1 \\ -8 & -8 & 2 \end{bmatrix}$.

2. 直接应用移位 QR 算法求以下矩阵的所有特征值:

(a) $\begin{bmatrix} 3 & 1 & 2 \\ 4 & 1 & 1 \\ -3 & 0 & 3 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 5 & 4 \\ 2 & -4 & -3 \\ 0 & -2 & 4 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & 1 & -2 \\ 4 & 2 & -3 \\ 0 & -2 & 2 \end{bmatrix}$;

(d) $\begin{bmatrix} 5 & -1 & 3 \\ 0 & 6 & 1 \\ 3 & 3 & -3 \end{bmatrix}$.

3. 直接应用移位 QR 算法求以下矩阵的全部特征值:

(a) $\begin{bmatrix} -1 & 1 & 3 \\ 3 & 3 & -2 \\ -5 & 2 & 7 \end{bmatrix}$; (b) $\begin{bmatrix} 7 & -33 & -15 \\ 2 & 26 & 7 \\ -4 & 50 & -13 \end{bmatrix}$; (c) $\begin{bmatrix} 8 & 0 & 5 \\ -5 & 3 & -5 \\ 10 & 0 & 13 \end{bmatrix}$;

(d) $\begin{bmatrix} -3 & 1 & 1 \\ 5 & 3 & -1 \\ -2 & 2 & 0 \end{bmatrix}$.

4. 重做计算机问题 3, 但是在应用 QR 迭代之前先化简为上 Hessenberg 形式. 打印 Hessenberg 形式及特征值.

5. 直接用 QR 算法, 求下面矩阵的所有的实的和复的特征值:

(a) $\begin{bmatrix} 4 & 3 & 1 \\ -5 & -3 & 0 \\ 3 & 3 & 1 \end{bmatrix}$; (b) $\begin{bmatrix} 3 & 2 & 0 \\ -4 & -2 & 1 \\ 2 & 1 & 0 \end{bmatrix}$; (c) $\begin{bmatrix} 7 & 2 & -4 \\ -8 & 0 & 7 \\ 2 & -1 & -2 \end{bmatrix}$;

(d) $\begin{bmatrix} 11 & 4 & -2 \\ -10 & 0 & 5 \\ 4 & 1 & 2 \end{bmatrix}$.

6. 用 QR 算法求特征值. 在每个矩阵中, 所有的特征值的绝对值 (或模) 相等, 所以可能需要 Hessenberg. 比较化简为 Hessenberg 形式之前和之后 QR 算法的结果.

(a) $\begin{bmatrix} -5 & -10 & -10 & 5 \\ 4 & 16 & 11 & -8 \\ 12 & 13 & 8 & -4 \\ 22 & 48 & 28 & -19 \end{bmatrix}$; (b) $\begin{bmatrix} 7 & 6 & 6 & -3 \\ -26 & -20 & -19 & 10 \\ 0 & -1 & 0 & 0 \\ -36 & -28 & -24 & 13 \end{bmatrix}$;

(c) $\begin{bmatrix} 13 & 10 & 10 & -5 \\ -20 & -16 & -15 & 8 \\ -12 & -9 & -8 & 4 \\ -30 & -24 & -20 & 11 \end{bmatrix}$.

实例检验 12 搜索引擎如何评定网页质量

像`Google.com`这样的网络搜索引擎是以搜索返回结果的质量来论优劣的. 我们来粗略讨论一下, Google 是怎样根据网络上存在的链接来评判网页的质量的.

当开始网络搜索, 搜索引擎要执行相当复杂的一系列任务. 一个明显的任务是逐词匹配, 即寻求标题或内容中包含查询词的网页. 另一个关键任务是评定所辨认出的网页, 帮助用户从大量的选择中找出自己的目标网页. 对于非常有针对性的查询, 可能只有几个主题匹配, 所有的结果都能返回到用户 (在网络发展的早期, 大家曾竟相去争取一矢中的). 这种情况下, 返回网页的质量不太重要, 因为结果不多, 甚至不需要对网页排序. 但对更一般的查询, 评定质量的需要就变得明显了. 例如在 Google 上查询 “新汽车” 会返回几百万网页, 开头的几乎都是汽车购买服务这种相当有用的输出. 这种评定是如何确定的?

这个问题的答案是, `Google.com` 对它所标的每一网页指定一个称为**页秩**(page rank) 的非负实数. 页秩由 Google 在世界上最大的持续进行的用于确定特征值的幂迭代中计算得到.

考虑如图 12-1 所示的图, 这里 n 个结点中的每一个都代表一网页, 并且从结点 i 到结点 j 的有向直线表示页面 i 包括了 j 的链接到页面. 设 $\boldsymbol{A}$ 表示**邻接矩阵**(adjacency matrix), 即 $n \times n$ 矩阵: 如果有结点 i 到结点 j 的链接, 那么它的第 ij 元素是 1; 否则是 0. 对于图 12-1, 邻接矩阵是

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

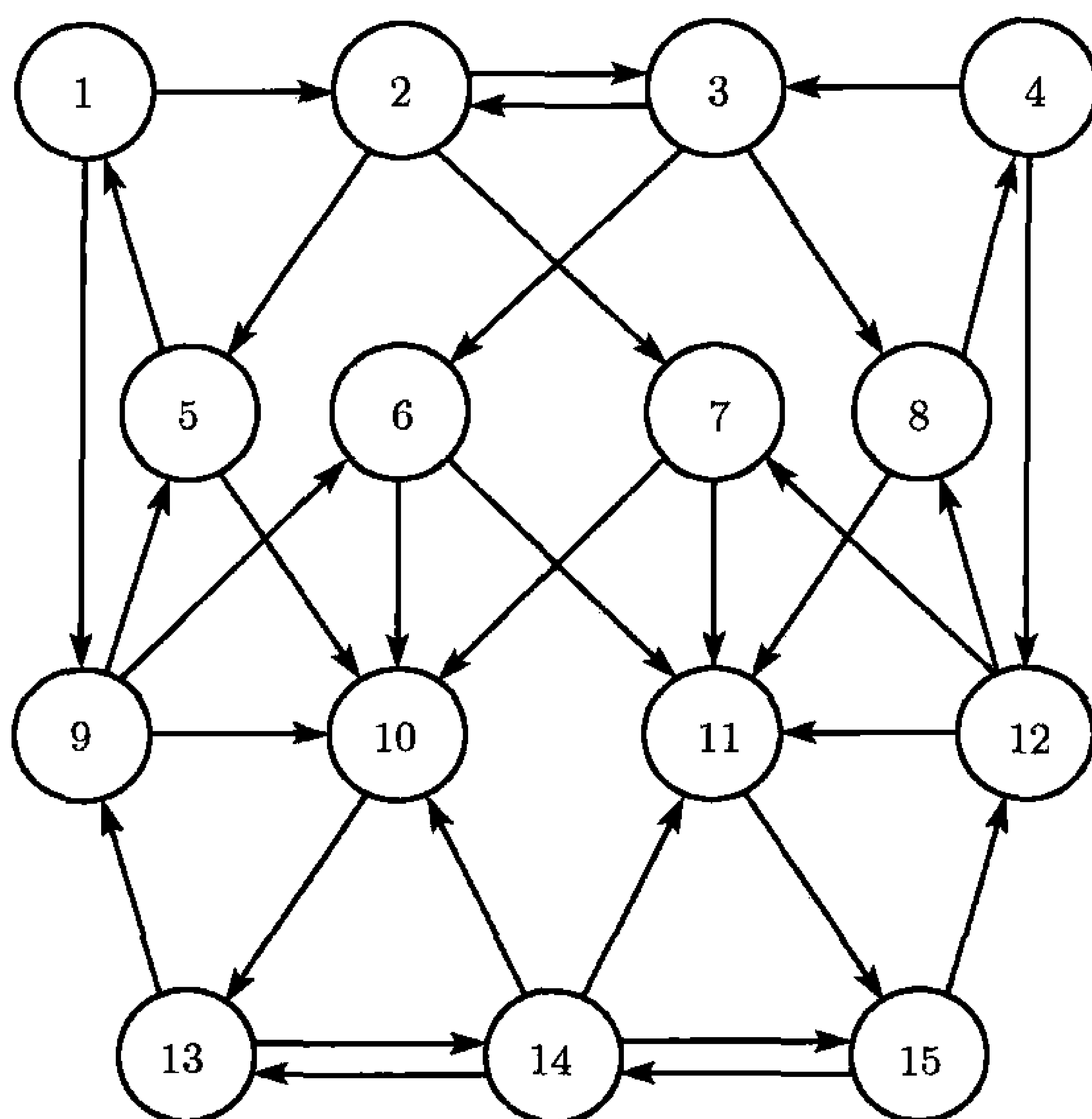


图 12-1 网页及链接的网络. 每一条从一页到另一页的有向边表示第一页包含至少一个第二页的链接.

Google 的创始人想像一个在 n 页网络上的冲浪者, 他目前停留在第 i 页的概率是 p_i, 下面这个冲浪者或者 (以固定的概率 q, 经常近似地等于 0.15) 向随机页运动, 或者以概率 $1-q$ 随机地单击目前第 i 页的链接. 在单击之后, 这个冲浪者从第 i 页运动到第 j 页的概率是 $q/n+(1-q)A_{ij}/n_i$, 这里 A_{ij} 是邻接矩阵 $\boldsymbol{A}$ 的元素, 而 n_i 是 $\boldsymbol{A}$ 的第 i 行之和 (实际上, 第 i 页的链接数).

时间是任意的, 因此人在结点 j 的概率是这个表达式取遍所有 i 的和, 并且与时间无关, 即

$$p_j=\sum_i\left[\frac{qp_i}{n}+(1-q)\frac{p_i}{n_i}A_{ij}\right),$$

用矩阵表示, 它等价于特征值方程

$$\boldsymbol{p}=\boldsymbol{G}\boldsymbol{p}, \tag{12.14}$$

这里 $p=(p_i)$ 是人在 n 个页面上的 n 个概率的向量, $\boldsymbol{G}$ 是其第 ij 元素为 $q/n+A_{ji}(1-q)n_j$ 的矩阵. 我们将把 $\boldsymbol{G}$ 称为**google 矩阵**. 矩阵 $\boldsymbol{G}$ 的每列之和为 1, 所以它是随机矩阵, 而且根据习题 12.2.4 它的最大特征值等于 1. 相应于特征值 1 的特征向量 p 是页面的稳态概率的集合. 按照定义它们是 n 个页面的页秩 (这是由 $\boldsymbol{G}^{\mathrm{T}}$ 定义的 Markov 过程的稳态解. 用稳态概率测量效应的最初思想追溯到 Pinski 和 Narin[8]. 跳变概率 (jump probability)q 是由 Brin 和 Page[3] 加上去的, 他们是 Google 的创始人).

我们将用如图 12-1 所示的例子来说明页秩的定义. 设 $q=0.15$, google 矩阵 $\boldsymbol{G}$ 的主特征向量 (相应于主特征值 1) 是

$$p = \begin{bmatrix} 0.026\ 8 \\ 0.029\ 9 \\ 0.029\ 9 \\ 0.026\ 8 \\ 0.039\ 6 \\ 0.039\ 6 \\ 0.039\ 6 \\ 0.039\ 6 \\ 0.074\ 6 \\ 0.106\ 3 \\ 0.106\ 3 \\ 0.074\ 6 \\ 0.125\ 1 \\ 0.116\ 2 \\ 0.125\ 1 \end{bmatrix}.$$

这个特征向量已经过除以各分量之和而标准化，就像概率所应该具备的那样，其和为 1. 结点 13 和结点 15 的页秩最高，接下来是结点 14, 10, 11. 注意页秩不是简单地取决于"内向级别"或者链接到这一页的内向点 (inward-pointing) 的个数，而是要靠分派重要性等更复杂的机制. 虽然结点 10 和结点 11 有最多的内向点链接，但是它们指向 13 和 15 这一事实却无形中抬高了别人. 这就是 "google 轰炸" 后面的思想，即通过使大流量站点链接到某一站点来人为地提高后者的重要性.

记住，用这种方法定义页秩时，我们用了"重要性"，尽管没有人真正知道它表示什么. 页秩是一种指定重要性的自我参考方法，在找到一种更好的方法之前它可能足够了.

建议习题

1. 证明 google 矩阵 G 是随机矩阵.
2. 对所示网络构造矩阵 G，并验证给定的主特征向量 p.
3. 把跳变概率 q 改变成 (a)0, (b)0.5. 叙述结果在页秩的改变. 跳变概率的作用是什么?
4. 假设网络中页面 7 想要改进它相对于其竞争者页面 6 的页秩，譬如说，通过促使页面 2 和页面 12 更显著地展示它们到页面 7 的链接. 通过在邻接矩阵中用 2 代替 A_{27} 和 $A_{12,7}$ 来模拟它. 这种策略是否成功?
5. 研究从网络移除页面 10 的影响 (删除所有到页面 10 和来自页面 10 的链接). 哪一个页秩增大了，哪一个减小了.
6. 设计你自己的网络，计算页秩并且按照前面的问题进行分析.

12.3 奇异值分解

R^m 中的单位球在 $m \times m$ 矩阵作用下的象是椭球. 这个有趣的事实构成奇异值分解的基础，它在一般的矩阵分析，特别是在有压缩效果的矩阵分析中有着许多

应用, 图 12-2 是对应于矩阵

$$A = \begin{bmatrix} 3 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \tag{12.15}$$

的椭圆的图解.

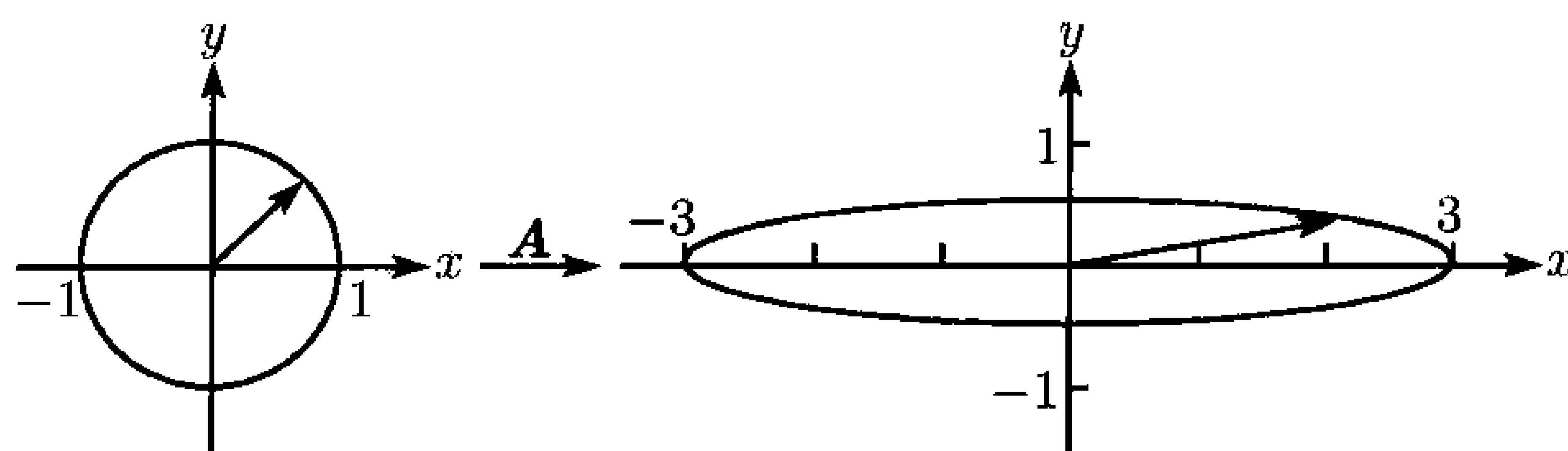


图 12-2　单位圆在 2×2 矩阵作用下的象. 在 R^2 中的单位圆通过 (12.15) 中的矩阵 A 被映射成半长轴为 (3,0) 和 $(0, \frac{1}{2})$ 的椭圆

在图 12-2 中, 想像取对应于单位圆上每一点的向量 v, 用 A 相乘然后画出结果向量 Av 的终点, 其结果就是所示的椭圆. 为了描述这个椭圆, 用标准正交向量集定义坐标系的基是有帮助的.

我们在定理 12.11 中将看到: 对每一个 $m \times n$ 矩阵 A, 存在标准正交集 $\{u_1, \cdots, u_m\}$, $\{v_1, \cdots, v_n\}$, 以及非负数 $s_1 \geqslant \cdots \geqslant s_n \geqslant 0$, 满足

$$\begin{aligned} Av_1 &= s_1 u_1, \\ Av_2 &= s_2 u_2, \\ &\vdots \\ Av_n &= s_n u_n. \end{aligned} \tag{12.16}$$

这些向量在图 12-3 中可以直观地看到. v_i 称为是矩阵 A 的**右奇异向量**, u_i 是 A 的**左奇异向量**, s_i 是 A 的**奇异值**(singular value). (关于这些向量的专门名词是有一点奇怪, 但是原因不久将会变得很清楚).

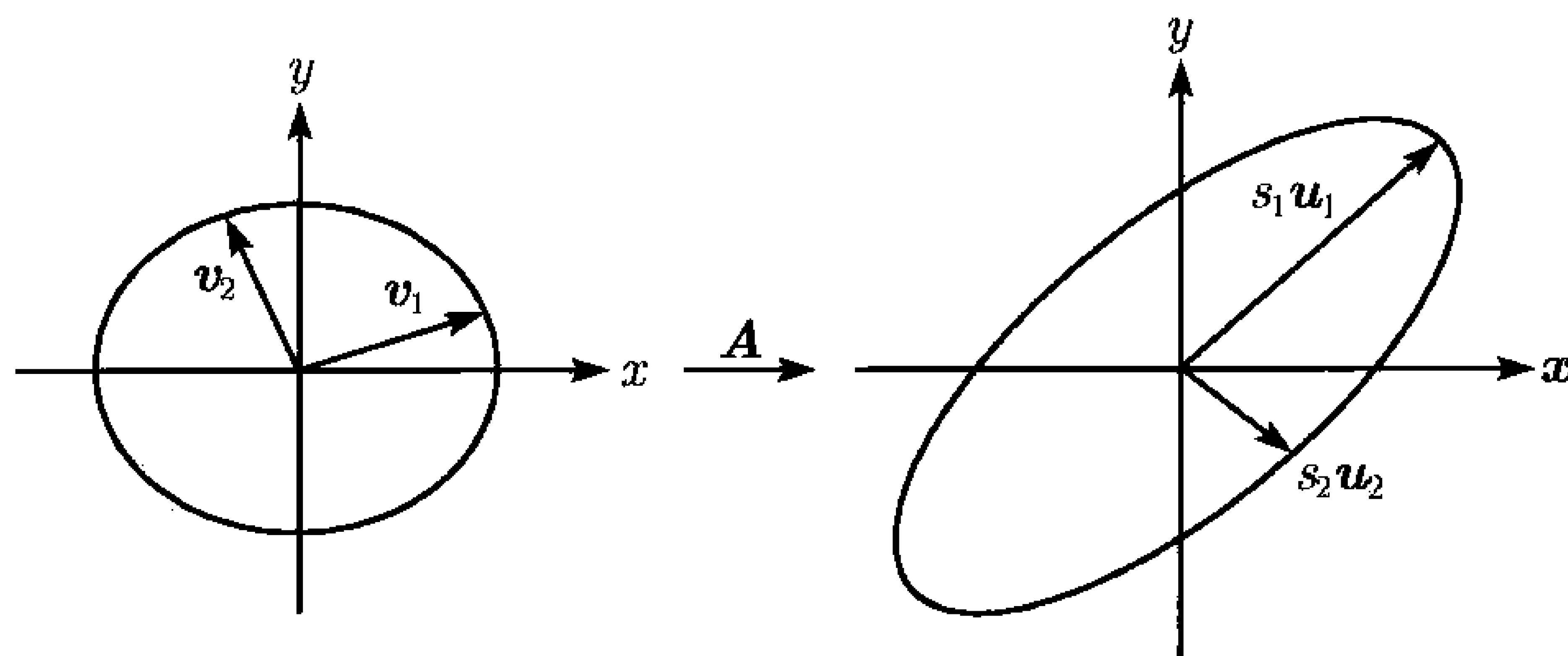


图 12-3　与矩阵相关的椭圆. 可以用以下简单方式来看待每一个 2×2 矩阵 A: 存在坐标系 $\{v_1, v_2\}$, 对于它 A 传递 $v_1 \to s_1 u_1$ 以及 $v_2 \to s_2 u_2$, 这里 $\{u_1, u_2\}$ 是另一个坐标系, 而 s_1, s_2 是非负数. 对 $m \times m$ 矩阵这个图推广到 R^m

这个有用的事实马上解释了为什么 2×2 矩阵把单位圆映射到椭圆. 我们可以认为 v_i 是直角坐标系的基. 而 $\boldsymbol{A}$ 以一种简单方式在其上作用: 它产生新坐标系的基向量 $\boldsymbol{u}_i$, 具有以标量 s_i 作为量化的伸长. 伸长的基向量 $s_i\boldsymbol{u}_i$ 是椭圆的半主轴, 如图 12-3 所示.

例 12.4 对于图 12-2 中表示的矩阵 (12.15), 求奇异值和奇异向量.

显然, 矩阵在 x 方向伸长到原来的 3 倍, 而在 y 方向缩小到原来的 $\frac{1}{2}$. $\boldsymbol{A}$ 的奇异向量和奇异值是

$$\boldsymbol{A}\begin{bmatrix}1\\0\end{bmatrix}=3\begin{bmatrix}1\\0\end{bmatrix},\qquad \boldsymbol{A}\begin{bmatrix}0\\1\end{bmatrix}=\frac{1}{2}\begin{bmatrix}0\\1\end{bmatrix}\tag{12.17}$$

向量 $3(1,0)$ 和 $\frac{1}{2}(0,1)$ 形成椭圆的半主轴. 右奇异向量是 $[1,0],[0,1]$, 左奇异向量是 $[1,0],[0,1]$. 奇异值是 3 和 $\frac{1}{2}$. ◀

例 12.5 求矩阵

$$\boldsymbol{A}=\begin{bmatrix}0&-\frac{1}{2}\\3&0\\0&0\end{bmatrix}\tag{12.18}$$

的奇异值和奇异向量.

这是对例 12.4 的一种小的改变. 矩阵以某种尺度的改变变换了 x 和 y 轴, 并且加上了 z 轴, 沿着 z 轴没有发生变化. $\boldsymbol{A}$ 的奇异向量和奇异值是

$$\begin{aligned}\boldsymbol{A}\boldsymbol{v}_1=\boldsymbol{A}\begin{bmatrix}1\\0\end{bmatrix}=3\begin{bmatrix}0\\1\\0\end{bmatrix}=s_1\boldsymbol{u}_1,\\ \boldsymbol{A}\boldsymbol{v}_2=\boldsymbol{A}\begin{bmatrix}0\\1\end{bmatrix}=\frac{1}{2}\begin{bmatrix}-1\\0\\0\end{bmatrix}=s_2\boldsymbol{u}_2\end{aligned}\tag{12.19}$$

右奇异向量是 $[1,0],[0,1]$, 而左奇异向量是 $[0,1,0],[-1,0,0]$, 奇异值是 $3,\frac{1}{2}$. 注意我们永远要求 s_i 是非负数, 任何必要的负号被吸收到 $\boldsymbol{u}_i$ 和 $\boldsymbol{v}_i$ 中去. ◀

在 $m\times n$ 矩阵 $\boldsymbol{A}$ 的矩阵分解中, 有一种记着这个信息的标准方法. 作出 $m\times m$ 矩阵 $\boldsymbol{U}$(它的列是左奇异向量 $\boldsymbol{u}_i$), $n\times n$ 矩阵 $\boldsymbol{V}$(它的列是右奇异向量 $\boldsymbol{v}_i$), 以及对角 $m\times n$ 矩阵 $\boldsymbol{S}$, 它的对角元素是奇异值 s_i. 那么 $m\times n$ 矩阵 $\boldsymbol{A}$ 的**奇异值分解**(SVD) 是

$$\boldsymbol{A}=\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}}.\tag{12.20}$$

例 12.5 有 SVD 表达式

$$\begin{bmatrix} 0 & -\frac{1}{2} \\ 3 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{12.21}$$

因为 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 是有正交列的方阵, 所以它们是正交矩阵. 注意我们已把第三列 $\boldsymbol{u}_3$ 加到 $\boldsymbol{U}$ 完成了 R^3 的基. 最后我们能够解释术语. $\boldsymbol{u}_i(\boldsymbol{v}_i)$ 是左 (右) 奇异向量, 这是因为它们出现在矩阵表达式 (12.21) 的那一边.

12.3.1 一般情况下求 SVD

我们已经展示了 SVD 的两个简单例子. 要说明对一般矩阵 $\boldsymbol{A}$, SVD 存在, 我们需要以下引理.

引理 12.9 *设 $\boldsymbol{A}$ 是一个 $m \times n$ 矩阵, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 的特征值非负.*

证 *设 $\boldsymbol{v}$ 是 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 的单位特征向量, 并且 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$, 那么*

$$0 \leqslant \|\boldsymbol{A}\boldsymbol{v}\|^2 = \boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v} = \lambda.$$

对于 $m \times n$ 矩阵 $\boldsymbol{A}$, $n \times n$ 矩阵 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 是对称的, 所以它的特征向量正交并且特征值是实数. 引理 12.9 说明其特征值是非负实数所以应该表示为 $s_1^2 \geqslant \cdots \geqslant s_n^2$, 这里相应的正交特征向量集是 $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$. 这已经给了我们 SVD 的 $\frac{2}{3}$. 用以下方法求 $\boldsymbol{u}_i, 1 \leqslant i \leqslant m$:

如果 $s_i \neq 0$, 用方程 $s_i\boldsymbol{u}_i = \boldsymbol{A}\boldsymbol{v}_i$ 定义 $\boldsymbol{u}_i$.

如果 $s_i = 0$, 选择 $\boldsymbol{u}_i$ 为满足与 $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{i-1}$ 正交的任意单位向量.

读者应该检验这种选择意味着 $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m$ 是两两正交的单位向量, 因此是 R^m 的另一个正交基. 事实上, $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m$ 形成 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ 的一组标准正交特征向量 (见习题 4). 总结一下, 我们已经证明了定理 12.10.

定理 12.10 *设 $\boldsymbol{A}$ 是 $m \times n$ 矩阵, 那么存在 R^n 和 R^m 的两个正交基 $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$, $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m\}$, 以及实数 $s_1 \geqslant \cdots \geqslant s_n \geqslant 0$ 使得对于 $1 \leqslant i \leqslant \min\{m, n\}$, $\boldsymbol{A}\boldsymbol{v}_i = s_i\boldsymbol{u}_i$.$\boldsymbol{V} = [\boldsymbol{v}_1|\cdots|\boldsymbol{v}_n]$ 的列, 即右奇异向量, 是 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 的正交特征向量组; $\boldsymbol{U} = [\boldsymbol{u}_1|\cdots|\boldsymbol{u}_m]$ 的列, 即左奇异向量, 是 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ 的正交特征向量组.*

对于给定矩阵 $\boldsymbol{A}$, SVD 不唯一. 例如在定义方程 $\boldsymbol{A}\boldsymbol{v}_1 = s_1\boldsymbol{u}_1$ 中, 用 $-\boldsymbol{v}_1$ 代替 $\boldsymbol{v}_1$, 用 $-\boldsymbol{u}_1$ 代替 $\boldsymbol{u}_1$ 不改变相等性, 但是改变矩阵 $\boldsymbol{U}$ 和 $\boldsymbol{V}$.

我们从这个定理可知单位球向量的象是中心在原点、半主轴为 $s_i\boldsymbol{u}_i$ 的椭球向量. 图 12-3 展示了单位圆向量被映射到轴为 $\{s_1\boldsymbol{u}_1, s_2\boldsymbol{u}_2\}$ 椭圆. 对于向量 $\boldsymbol{x}$, 为了求 $\boldsymbol{A}\boldsymbol{x}$ 去向何处, 我们可以写 $\boldsymbol{x} = a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2$[这里 $a_1\boldsymbol{v}_1(a_2\boldsymbol{v}_2)$ 是 $\boldsymbol{x}$ 在 $\boldsymbol{v}_1(\boldsymbol{v}_2)$ 方向的投影], 那么 $\boldsymbol{A}\boldsymbol{x} = a_1s_1\boldsymbol{u}_1 + a_2s_2\boldsymbol{u}_2$.

矩阵表达式 (12.20) 直接从定理 12.11 得出. 定义 $\boldsymbol{S}$ 是 $m \times n$ 对角矩阵, 它的元素 $s_1 \geqslant \cdots \geqslant s_{\min\{m,n\}} \geqslant 0$. 定义 $\boldsymbol{U}$ 是列为 $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m$ 的矩阵, $\boldsymbol{V}$ 是列为 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ 的矩阵. 注意对 $i = 1, \cdots, m$, $\boldsymbol{USV}^{\mathrm{T}}\boldsymbol{v}_i = s_i\boldsymbol{u}_i$. 因为矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{USV}^{\mathrm{T}}$ 关于基 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ 一致, 所以它们是相同的 $m \times n$ 矩阵.

例 12.6 求 2×2 矩阵

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \tag{12.22}$$

的奇异值和奇异向量.

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$$

的特征值是 $\boldsymbol{v}_1 = (0,1), s_1^2 = 2$ 以及 $\boldsymbol{v}_2 = (1,0), s_2^2 = 0$. 奇异值是 $\sqrt{2}$ 以及 0. 根据前面的方法, $\boldsymbol{u}_1$ 定义为

$$\sqrt{2}\boldsymbol{u}_1 = \boldsymbol{A}\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad \boldsymbol{u}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix},$$

并且选择 $\boldsymbol{u}_2 = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$ 与 $\boldsymbol{u}_1$ 正交. SVD 是

$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \sqrt{2} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{12.23}$$

依据定理 12.10 后面的不唯一性的说明, 关于这个矩阵的另一完美的 SVD 是

$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \sqrt{2} & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{12.24}$$

单位圆在 $\boldsymbol{A}$ 之下的象是线段 $y[1,-1]$, 这里 y 从 -1 延伸到 1. 所以 $\boldsymbol{A}$ 的作用是把单位圆压扁为半主轴为 $\sqrt{2}\left[\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right]$ 及 0 的一维椭圆. ◀

对于奇异值分解的 MATLAB命令是svd, 所以

```
>> [u,v,v]=svd(a)
```

将返回分解得到的所有 3 个矩阵.

12.3.2 特殊情形: 对称矩阵

求 $m \times n$ 对称矩阵的 SVD 只是简单地求解特征值和特征向量. 附录 A 中的定理 A.5 保证了存在一组正交特征向量. 因为特征向量映射到它们自己 (带上比例 λ, 它是特征值), 所以满足方程组 (12.16) 是容易的. 把特征值按度量大小递减排序为

$$|\lambda_1| \geqslant |\lambda_2| \geqslant |\lambda_3| \geqslant \cdots \geqslant |\lambda_m|, \tag{12.25}$$

并把它们当作奇异值 $s_1 \geqslant s_2 \geqslant \cdots$ 看待. 对于 $\boldsymbol{v}_i$, 以对应于 (12.25) 中特征向量的次序使用单位向量, 并且用

$$\boldsymbol{u}_i = \begin{cases} +\boldsymbol{v}_i, & 如果\lambda_i \geqslant 0, \\ -\boldsymbol{v}_i, & 如果\lambda_i < 0. \end{cases} \tag{12.26}$$

在 (12.26) 式中符号的改变弥补了在 (12.25) 中取绝对值而失去的任何负号.

例 12.7 求

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix} \tag{12.27}$$

的奇异值和奇异向量.

特征值/特征向量对是 $2, [1,2]^{\mathrm{T}}$ 以及 $-\frac{1}{2}, [-2,1]^{\mathrm{T}}$. 我们从单位特征向量定义 $\boldsymbol{v}_i$ 并且从 (12.26) 定义 $\boldsymbol{u}_i$:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{v}_1 &= \boldsymbol{A}\begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix} = 2\begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix} = s_1\boldsymbol{u}_1, \\ \boldsymbol{A}\boldsymbol{v}_2 &= \boldsymbol{A}\begin{bmatrix} \frac{2}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -\frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix} = s_2\boldsymbol{u}_2, \end{aligned} \tag{12.28}$$

SVD 是

$$\begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}. \tag{12.29}$$

注意如 (12.26) 中规定的, 我们已经改变符号以定义 $\boldsymbol{u}_2$. ◀

习题 12.3

1. 手工计算以下对称矩阵的 SVD, 并且在几何上叙述矩阵对单位圆的作用:

(a) $\begin{bmatrix} -3 & 0 \\ 0 & 2 \end{bmatrix}$; (b) $\begin{bmatrix} 0 & 0 \\ 0 & 3 \end{bmatrix}$; (c) $\begin{bmatrix} \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} \end{bmatrix}$; (d) $\begin{bmatrix} -\frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{3}{2} \end{bmatrix}$;

(e) $\begin{bmatrix} 0.75 & 1.25 \\ 1.25 & 0.75 \end{bmatrix}$.

2. 手工计算以下矩阵的 SVD:

(a) $\begin{bmatrix} 3 & 0 \\ 4 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 6 & -2 \\ 8 & \frac{3}{2} \end{bmatrix}$; (c) $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$; (d) $\begin{bmatrix} -4 & -12 \\ 12 & 11 \end{bmatrix}$;

(e) $\begin{bmatrix} 0 & -2 \\ -1 & 0 \end{bmatrix}$.

3. SVD 不是唯一的. 对例 12.4, 存在多少种不同的 SVD, 把它们列出来.

4. (a)证明在定理 12.11 中定义的 $\boldsymbol{u}_i$ 是 $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}$ 的特征向量. (b) 证明 $\boldsymbol{u}_i$ 是单位向量. (c) 证明它们形成 R^m 的一组正交基.

12.4 SVD 的应用

本节收集 SVD 的某些有用性质而且指出它们的某些广泛应用. 例如, SVD 被证实是求矩阵的秩的最好方法. 方阵的行列式以及逆 (如果它存在) 能从 SVD 求出. 或许 SVD 的最有用的应用是从低秩近似性质得出的.

12.4.1 SVD 的性质

$m\times n$ 矩阵 $\boldsymbol{A}$ 的**秩**(rank) 是它的线性无关的行 (或者等价地, 列) 的个数.

性质 1 矩阵 $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}}$ 的秩是 $\boldsymbol{S}$ 中非零元素的个数.

证 因为 $\boldsymbol{U}$ 和 $\boldsymbol{V}^{\mathrm{T}}$ 是可逆矩阵, 所以 $\mathrm{rank}(\boldsymbol{A})=\mathrm{rank}(\boldsymbol{S})$, 而后者就是非零对角元素的个数.

性质 2 如果 $\boldsymbol{A}$ 是 $n\times n$ 矩阵, 那么 $|\det(\boldsymbol{A})|=s_1\cdots s_n$.

证 因为 $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U}=\boldsymbol{I}$ 以及 $\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}=\boldsymbol{I}$, 所以 $\boldsymbol{U}$ 和 $\boldsymbol{V}^{\mathrm{T}}$ 的行列式是 1 或者 -1, 这是因为乘积的行列式等于行列式的乘积. 还是因为乘积的行列式是行列式的乘积, 从分解式 $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}}$ 就得到了性质 2.

性质 3 如果 $\boldsymbol{A}$ 是可逆 $m\times m$ 逆阵, 那么 $\boldsymbol{A}^{-1}=\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^{\mathrm{T}}$.

证 根据性质 1, $\boldsymbol{S}$ 可逆意味着所有的 $s_i>0$. 现在性质 3 从以下事实得出: 即如 $\boldsymbol{A}_1,\boldsymbol{A}_2$ 及 $\boldsymbol{A}_3$ 是可逆矩阵, 那么 $(\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3)^{-1}=\boldsymbol{A}_3^{-1}\boldsymbol{A}_2^{-1}\boldsymbol{A}_1^{-1}$.

例如, 来自 (12.29) 式的 SVD

$$\begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix}=\begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}$$

说明逆矩阵是

$$\begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix}^{-1}=\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}=\begin{bmatrix} -\frac{3}{2} & 1 \\ 1 & 0 \end{bmatrix}. \tag{12.30}$$

性质 4 $m\times n$ 矩阵 $\boldsymbol{A}$ 可以写成秩 1 矩阵之和:

$$\boldsymbol{A}=\sum_{i=1}^{r}s_i\boldsymbol{u}_i\boldsymbol{v}_i^{\mathrm{T}}, \tag{12.31}$$

这里 r 是 $\boldsymbol{A}$ 的秩, 而 $\boldsymbol{u}_i$ 及 $\boldsymbol{v}_i$ 分别是 $\boldsymbol{U}$ 及 $\boldsymbol{V}$ 的第 i 列.

证

$$
\begin{aligned}
\boldsymbol{A}=\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}}=\boldsymbol{U}\begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_r \end{bmatrix}\boldsymbol{V}^{\mathrm{T}} \\
=\boldsymbol{U}\left(\begin{bmatrix} s_1 & & \\ & & \\ & & \end{bmatrix}+\begin{bmatrix} & & \\ & s_2 & \\ & & \end{bmatrix}+\cdots+\begin{bmatrix} & & \\ & & \\ & & s_r \end{bmatrix}\right)\boldsymbol{V}^{\mathrm{T}} \\
=s_1\boldsymbol{u}_1\boldsymbol{v}_1^{\mathrm{T}}+s_2\boldsymbol{u}_2\boldsymbol{v}_2^{\mathrm{T}}+\cdots+s_r\boldsymbol{u}_r\boldsymbol{v}_r^{\mathrm{T}}.
\end{aligned}
$$

性质 4 是 SVD 的低秩近似性质. 通过保留 (12.31) 式的前面 p 项就提供了对秩 $p\leqslant r$ 的 $\boldsymbol{A}$ 的最佳最小二乘近似.

例 12.8 求矩阵 $\begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix}$ 的最佳秩 1 近似.

写出 (12.31) 就得到

$$
\begin{aligned}
\begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix} &= \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}\left(\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}+\begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\right)\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \\
&= 2\begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}+\frac{1}{2}\begin{bmatrix} -\frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix}\begin{bmatrix} \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{8}{5} \end{bmatrix}+\begin{bmatrix} -\frac{2}{5} & \frac{1}{5} \\ \frac{1}{5} & -\frac{1}{10} \end{bmatrix}.
\end{aligned}
\tag{12.32}
$$

因为奇异值的大小不同, 注意原矩阵是如何分成较大的基值加上较小的基值. 矩阵的最佳秩 1 近似由第一个秩 1 矩阵 $\begin{bmatrix} \frac{2}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{8}{5} \end{bmatrix}$ 给出. 而第二个矩阵提供了小的校正. 这是隐藏在 SVD 的降维和压缩应用背后的主要思想. ◀

下面两节与 SVD 有紧密相关的应用. 在降维中, 中心是用一批张成较低维的向量来近似一大批多维向量. 其他的应用是有损压缩, 把近似地表示成一个矩阵需要的信息量压缩. 这两种应用都取决于关于低秩近似的性质 4.

12.4.2 降维

想法是把数据投影到较低的维数. 假设 $a_1,\cdots,a_n$ 构成了一批 m 维向量. 在资料丰富的应用中, m 远比 n 小. 降维的目标是用张成 $p<m$ 维的 n 个向量来代

替 $a_1, \cdots, a_n$, 同时使与这样做相关的误差极小化. 通常我们从平均为零的一组向量开始. 否则, 我们可以减去这个平均向量而达到这个要求. 并且在后来再把它加起来.

SVD 给出了一个直接了当的方法执行降维. 把数据向量考虑为 $m \times n$ 矩阵 $\boldsymbol{A} = (\boldsymbol{a}_1|\cdots|\boldsymbol{a}_n)$ 的列, 并且计算奇异值分解 $\boldsymbol{A} = \boldsymbol{USV}^{\mathrm{T}}$, 设 $\boldsymbol{e}_j$ 表示第 j 个基本基向量 (即第 j 个分量是 1, 其他分量全为 0). 于是 $\boldsymbol{Ae}_j = \boldsymbol{a}_j$. 利用性质 4 的秩 p 近似

$$\boldsymbol{A} \approx \boldsymbol{A}_p = \sum_{i=1}^{p} s_i \boldsymbol{u}_i \boldsymbol{v}_i^{\mathrm{T}},$$

我们可以通过

$$\boldsymbol{a}_j = \boldsymbol{Ae}_j \approx \boldsymbol{A}_p \boldsymbol{e}_j \tag{13.33}$$

把 $\boldsymbol{a}_j$ 投影到由 $\boldsymbol{U}$ 的列 $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_p$ 所张成的 p 维空间中, 因为把一个矩阵乘以 $\boldsymbol{e}_j$ 恰好是把它的第 j 列挑选出来, 我们能更有效地叙述求解如下:

由左奇异向量 $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_p$ 张成的空间 $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_p\}$ 是在最小二乘意义下对 $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n$ 的最佳近似 p 维子空间, 而且 $\boldsymbol{A}$ 的列 $\boldsymbol{a}_i$ 到这个空间的正交投影是 $\boldsymbol{A}_p$ 的列. 换言之, 一批向量 $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n$ 到它们的最佳最小二乘 p 维子空间的投影正好就是最佳秩 p 近似矩阵 $\boldsymbol{A}_p$.

例 12.9 求拟合数据向量 $[3,2], [2,4], [-2,-1], [-3,-5]$ 的最佳一维子空间.

图 12-4a 所示的 4 个数据向量近似地指向相同的一维子空间. 我们想要求这个子空间, 它把这些向量投影到这个子空间所产生的误差的平方之和极小化, 然后求这些投影向量.

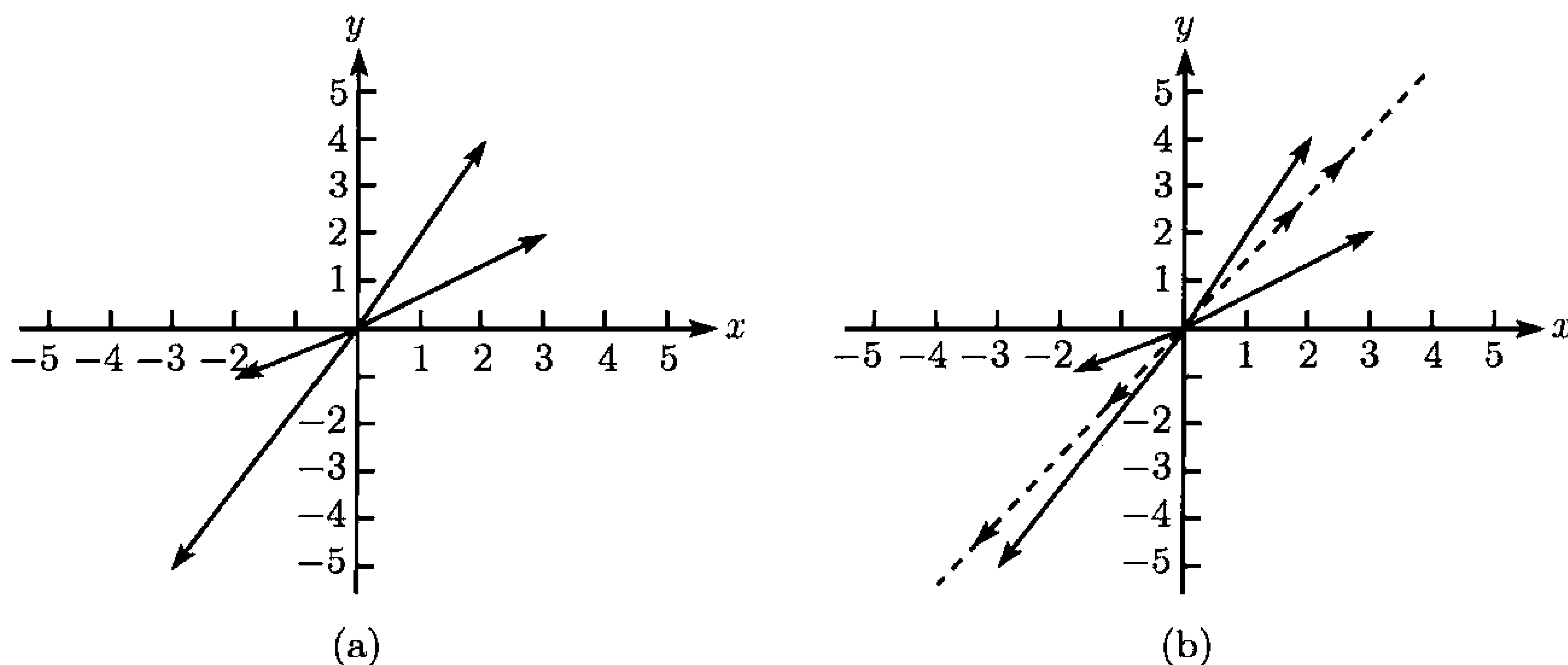


图 12-4 用 SVD 降维. (a) 4 个数据向量被投影到最佳一维子空间. (b) 虚线表示最佳子空间. 箭头表明到子空间的正交投影

把数据向量用作数据矩阵

$$A = \begin{bmatrix} 3 & 2 & -2 & -3 \\ 2 & 4 & -1 & -5 \end{bmatrix}$$

的列并且求它的 SVD, 精确到 4 位小数它是

$$\begin{bmatrix} 0.588\,6 & -0.808\,4 \\ 0.808\,4 & 0.588\,6 \end{bmatrix} \begin{bmatrix} 8.280\,9 & 0 & 0 & 0 \\ 0 & 1.851\,2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.408\,5 & 0.532\,7 & -0.239\,8 & -0.701\,4 \\ -0.674\,1 & 0.398\,5 & 0.555\,4 & -0.279\,8 \\ 0.574\,3 & -0.189\,2 & 0.792\,4 & -0.080\,1 \\ 0.221\,2 & 0.722\,3 & 0.078\,0 & 0.650\,7 \end{bmatrix}.$$

在图 12-4b 中以虚线表示的最佳一维子空间是由 $u_1 = [0.588\,6, 0.808\,4]$ 张成的. 降到 $p = 1$ 维的子空间意味着置 $s_2 = 0$ 并重新构造矩阵, 换言之 $\boldsymbol{A}_1 = \boldsymbol{U}\boldsymbol{S}_1\boldsymbol{V}^{\mathrm{T}}$, 这里

$$\boldsymbol{S}_1 = \begin{bmatrix} 8.280\,9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

因此

$$\boldsymbol{A}_1 = \begin{bmatrix} 1.991\,2 & 2.596\,4 & -1.168\,9 & -3.418\,8 \\ 2.734\,6 & 3.565\,7 & -1.605\,2 & -4.695 \end{bmatrix} \tag{12.34}$$

的列是对应于原来的 4 个数据向量的 4 个投影向量. 它们如图 12-4b 所示. ◀

12.4.3 压缩

性质 4 也能被用来压缩矩阵的信息. 注意在性质 4 的秩 1 展开式中的每一项用两个向量 $\boldsymbol{u}_i, \boldsymbol{v}_i$ 以及还有一个数 s_i 表示. 如果 $\boldsymbol{A}$ 是 $n \times n$ 矩阵, 我们可以通过把性质 4 中和的最后几项 (这些项带有较小的 s_i) 去掉来尝试有损压缩. 在这个展开式中的每一项需要 $2n+1$ 个数存储或传送.

例如, 如果 $n = 8$, 矩阵由 64 个数确定. 但是我们可以仅仅用 $2n+1 = 17$ 个数来传送或存储展开式中的第一项. 如果绝大部分信息由第一项截获——例如, 如果第一个奇异值比其余的大得多——用这种方式大约可以节约 75% 的空间.

作为例子, 回到图 11-6 所表示的 8×8 像素块. 减去 128 后使像素值集中于零周围, 矩阵由等式 (11.15) 给出. 这个 8×8 矩阵的奇异值如下:

$$\begin{array}{cccc} 485.52 & 364.33 & 64.55 & 60.91 \\ 18.98 & 10.24 & 7.31 & 2.40 \end{array}$$

原来的块如图 12-5a 所示, 压缩形式如图 12-5b 和图 12-5c 所示. 图 12-5b 相应于用性质 4 的展开式中的第一项代替这个矩阵, 是像素值矩阵的最佳秩 1 近似. 如前

所述, 它近似地达到 4:1 压缩. 在图 12-5c 中, 用了两项, 达到 2:1 的近似压缩比 (当然, 通过不用量化策略, 我们在这里简化了讨论. 如在第 11 章所做的, 它将以较低的精度帮助把系数与较小的奇异值相对应).

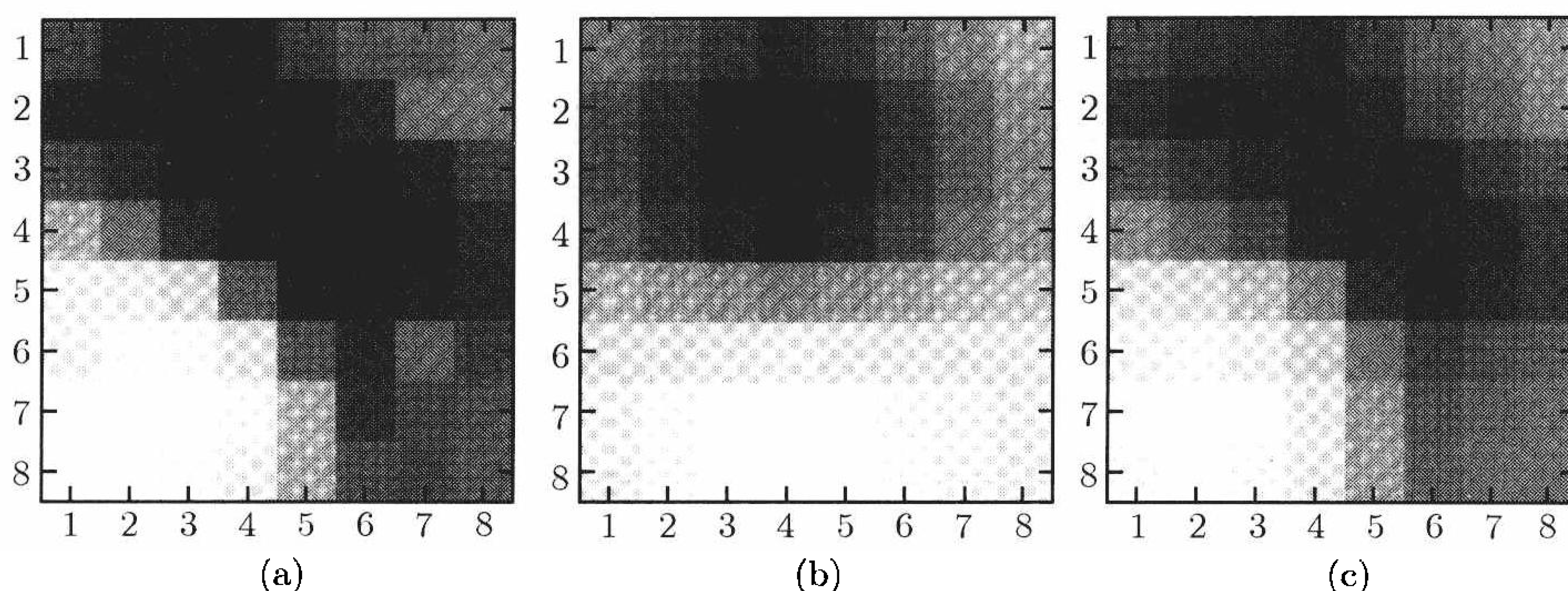


图 12-5 用 SVD 进行压缩和解压的结果. 保持的奇异值个数: (a) 全部, (b)$p=1$, (c) $p=2$

图 12-5 中灰 (色标) 度照片是 256×256 像素图像在对每一个像素元素减去 128 后, 我们也可以把性质 4 应用到整个矩阵. 矩阵的 256 个奇异值从 9 637 到 2×10^{-13} 变化. 图 12-6 表示在性质 4 的秩 1 展开式中由于保持 p 项而得到的重构象. 对于 $p=8$, 仅仅 $8(2(256)+1)=4\ 104$ 个数需要存储. 与 $5(256)^2=65\ 536$ 个原来的像素值比较, 大约是 16:1 的压缩比. 在图 12-6c 中保持了 32 项, 压缩比近似于 4:1.

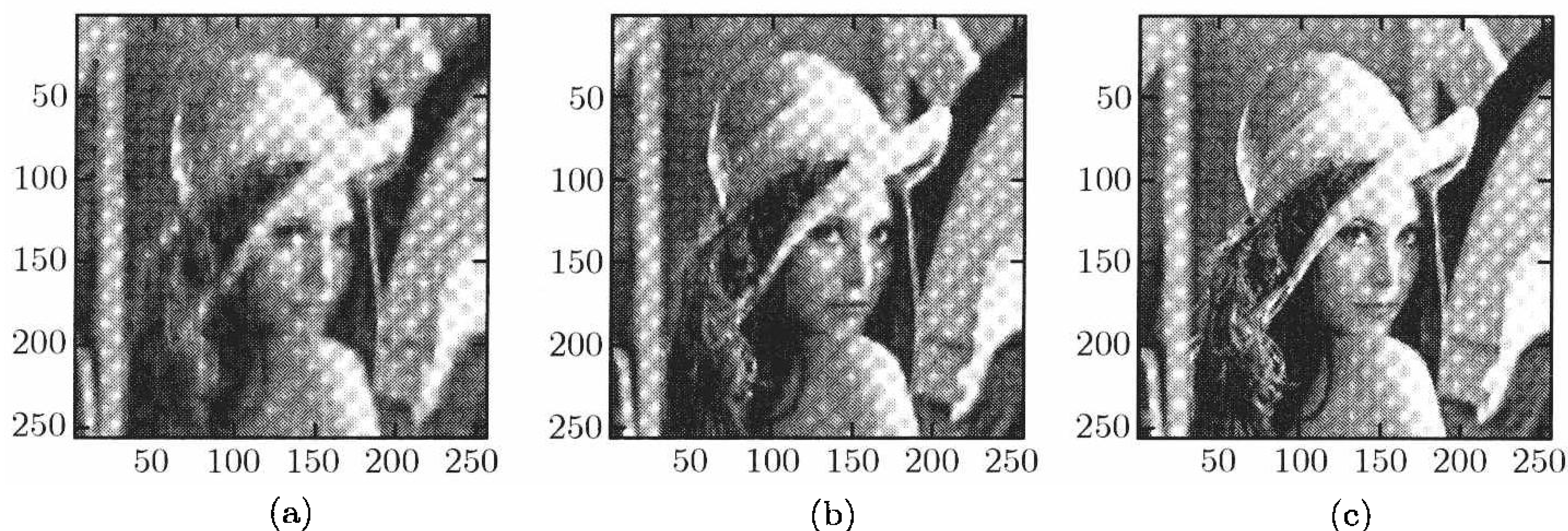


图 12-6 用 SVD 进行压缩和解压的结果. 保持的奇异值个数:(a) $p=8$, (b) $p=16$, (c) $p=32$

12.4.4 计算 SVD

如果 $\boldsymbol{A}$ 是实对称矩阵, SVD 化简为本章前面讨论过的特征值计算. 此时, 单位特征向量形成一组正交基. 如果我们定义矩阵 $\boldsymbol{V}$ 以单位特征向量为列, 那么 $\boldsymbol{AV}=\boldsymbol{US}$ 表示特征向量方程, 这里 $\boldsymbol{S}$ 是以特征值的绝对值为对角元的对角矩阵, $\boldsymbol{U}$ 与

V 除了像在 (12.26) 中讨论过的当特征值是负数时列的符号要改变外, 它们是相同的. 因为 U 和 V 是正交矩阵, 所以

$$A = USV^{\mathrm{T}}$$

是 A 的奇异值分解.

对于一般的, 非对称的 $m \times n$ 矩阵 A, 为了确定 SVD 有两种不同的计算方法. 第一种且非常明显的方法是形成 $A^{\mathrm{T}}A$ 并且求它的特征值. 根据定理 12.10, 它揭示了 V 的各列 v_i, 而且通过把向量 $Av_i = s_i u_i$ 标准化, 我们既得到了奇异值又得到了 U 的各列.

然而, 对于简单例子并不推荐这种方法. 如果 A 的条件数大, 那么 $A^{\mathrm{T}}A$ 的条件数 (经常是 A 的条件数的平方量级) 可能变得过分大, 因此可能损失精确的数字.

幸运的是, 有另外一种避免形成矩阵乘积的求 $A^{\mathrm{T}}A$ 的特征值的方法. 考虑矩阵

$$B = \begin{bmatrix} 0 & A^{\mathrm{T}} \\ A & 0 \end{bmatrix}. \tag{12.35}$$

注意 B 是对称的 $(m+n) \times (m+n)$ 矩阵 (检查它的转置). 因此, 它有非负实特征值以及一组特征向量基. 设 $[v, w]$ 表示 $(m+n)$ 维向量, 它是 B 的特征向量, 那么

$$\begin{bmatrix} A^{\mathrm{T}}w \\ Av \end{bmatrix} = \begin{bmatrix} 0 & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \lambda \begin{bmatrix} v \\ w \end{bmatrix},$$

或者 $Av = \lambda w$. 在左边乘以 A^{T} 得到

$$A^{\mathrm{T}}Av = \lambda A^{\mathrm{T}}w = \lambda^2 v, \tag{12.36}$$

表明 v 是 $A^{\mathrm{T}}A$ 的特征向量, 相应的特征值是 λ^2. 注意我们能够用这种方法确定 $A^{\mathrm{T}}A$ 的特征值和特征向量, 始终不需要形成矩阵 $A^{\mathrm{T}}A$.

因此, 计算奇异值和奇异向量的第二种更可取的方法从把对称矩阵 B 转移到上 Hessenberg 形式. 因为对称性, 上 Hessenberg 形式等价于三对角矩阵. 于是像移位 QR 算法这样的方法可以用于求等于奇异值平方的特征值, 以及其顶部的 n 个元素是奇异向量 v_i 的特征向量. 尽管这种方法把矩阵的大小加倍, 但是它避免了不必要的条件数的增加. 还有更有效的方法执行这种思想 (我们将不在这里进行), 它们不需要额外的存储.

计算机问题 12.4

1. 用 MATLAB的**svd**命令求下列矩阵的最佳秩 1 近似:

(a) $\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 3 \\ 0 & 0 & 1 \end{bmatrix}$; (d) $\begin{bmatrix} 1 & 5 & 3 \\ 2 & -3 & 2 \\ -3 & 1 & 1 \end{bmatrix}$.

2. 求下列矩阵的最佳秩 2 近似:

(a) $\begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 3 \\ 0 & 0 & 1 \end{bmatrix}$; (b) $\begin{bmatrix} 2 & -2 & 4 \\ 1 & -1 & 2 \\ -3 & 3 & -6 \end{bmatrix}$; (c) $\begin{bmatrix} 1 & 5 & 3 \\ 2 & -3 & 2 \\ -3 & 1 & 1 \end{bmatrix}$.

3. 求下列向量的最佳最小二乘近似直线以及这些向量到一维子空间的投影:

(a) $\begin{bmatrix} 1 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 5 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \end{bmatrix}$; (b) $\begin{bmatrix} 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \end{bmatrix}$;

(c) $\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 6 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$.

4. 求下列三维向量的最佳最小二乘近似平面, 以及这些向量到这个子空间的投影:

(a) $\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 6 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$; (b) $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 4 \\ 0 \end{bmatrix}, \begin{bmatrix} 7 \\ -2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$.

5. 写出用矩阵 (12.35) 计算矩阵奇异值的 MATLAB程序. 用以前给出的上 Hessenberg 代码, 并且用移位 QR 求最后得到的特征值问题. 用你的方法求下列矩阵的奇异值:

(a) $\begin{bmatrix} 3 & 0 \\ 4 & 0 \end{bmatrix}$; (b) $\begin{bmatrix} 6 & -2 \\ 8 & \frac{3}{2} \end{bmatrix}$; (c) $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$; (d) $\begin{bmatrix} -4 & -12 \\ 12 & 11 \end{bmatrix}$;

(e) $\begin{bmatrix} 0 & -2 \\ -1 & 0 \end{bmatrix}$.

6. 继续计算机问题 5, 加上代码求矩阵的完整的 SVD.

7. 使用在计算机问题 6 中建立的代码求以下矩阵的完整的 SVD, 并把你的结果与 MATLAB命令`svd`比较 (你的答案应与在 $\boldsymbol{u}_i, \boldsymbol{v}_i$ 中负号的选择相一致):

(a) $\begin{bmatrix} 1 & 3 & 0 \\ 4 & 5 & 0 \\ 2 & 5 & 3 \end{bmatrix}$; (b) $\begin{bmatrix} 1 & 0 & 2 & 4 \\ 1 & 1 & 1 & 3 \end{bmatrix}$; (c) $\begin{bmatrix} 0 & 1 & 3 \\ 1 & 3 & 1 \\ 2 & -1 & 3 \\ 0 & 1 & -1 \end{bmatrix}$;

(d) $\begin{bmatrix} 0 & 1 & 3 & 1 \\ -1 & 1 & 1 & 0 \\ 0 & 1 & 3 & -1 \\ 2 & -1 & -1 & 2 \end{bmatrix}$.

8. 用 MATLAB的`imread`命令导入相片. 用 SVD 来创建相片的 8 : 1, 4 : 1 及 2 : 1 压缩形

式. 如果相片是彩色的, 分别压缩每一种 RGB 色彩.

软件和进一步阅读

特征值计算的现代阶段是由 Wilkinson 的教科书 [13] 开创的,QR 算法及上 Hessenberg 形式已经出现在 [14]. 关于特征值计算有影响的参考书是 [11,6,5] 以及提示文章 [7,12].

Lapack[1] 对化简上 Hessenberg 形式和对称及非对称特征值问题提供了程序. 这些程序从 20 世纪 60 年代建立的 Eispach 软件包 [10] 转变而来. Netlib 的 DGEHRD 通过使用 Householder 反射把实矩阵化简为上 Hessenberg 形式, 并且 DHSEQR 实现了用于计算特征值的 QR 算法以及用于实上 Hessenberg 矩阵的 Schur 形式. NAG 对同样的两种运算分别提供了 F08NEF 和 F08PEF. 对复矩阵有类似的程序.

教科书 [9,2] 对大型特征值问题考虑了最新方法. Cuppen[4] 对三对角对称特征值问题引进了分治方法. Arpack 适于大型稀疏问题的 Arnoldi 迭代, Parpack 是对并行处理器的一种扩展.

奇异值分解的算法包括 Lapack 的 DGESVE 以及对大型矩阵更可取的分治方法 DGESDD. 复数形式也是适用的.

第13章　最　优　化

从 1953 年 DNA 双螺旋结构的发现开始, 半个世纪以来, 已经发现了人类基因组几乎全部的排序. 这个序列持有把氨基酸串与保持生命活动的单个蛋白质混合起来的指令, 但却写成加密语言形式. 该信息现在等待翻译, 通过它人们可以详尽地了解它的生理机能. 包括基因疗法和药物设计在内的一系列潜在应用, 将促进疾病的早期预防、诊断和治愈.

把氨基酸到功能蛋白质结合起来关键依靠范德华 (Van der Waals) 力, 即自由原子之间的微观吸引力和排斥力. 原子群模型, 其中这些力通过 Lennard-Jones 势能建模, 用于最小化能量构造的研究, 它将问题引入了最优化领域.

实例检验　13.2.3 节后的实例检验 13 应用本章的优化方法去求解能源最小化问题.

最优化是指找一个实值函数即**目标函数**(objective function) 的最大值或最小值. 由于求解函数 $f(x)$ 的最大值等价于求函数 $-f(x)$ 的最小值, 因此在设计计算方法的时候只需考虑最小化就可以了.

一些最优化问题需要求带有几个等式和不等式约束的目标函数的最小值. 例如, 尽管 x_1 是图 13-1 中函数的全局最小值, 但 x_2 却是满足约束条件 $x \geqslant 0$ 的最小值. 特别地, 线性规划考虑目标函数和约束条件都是线性的问题. 在本章中, 我们将继续使事情简化, 仅仅考虑无约束最优化.

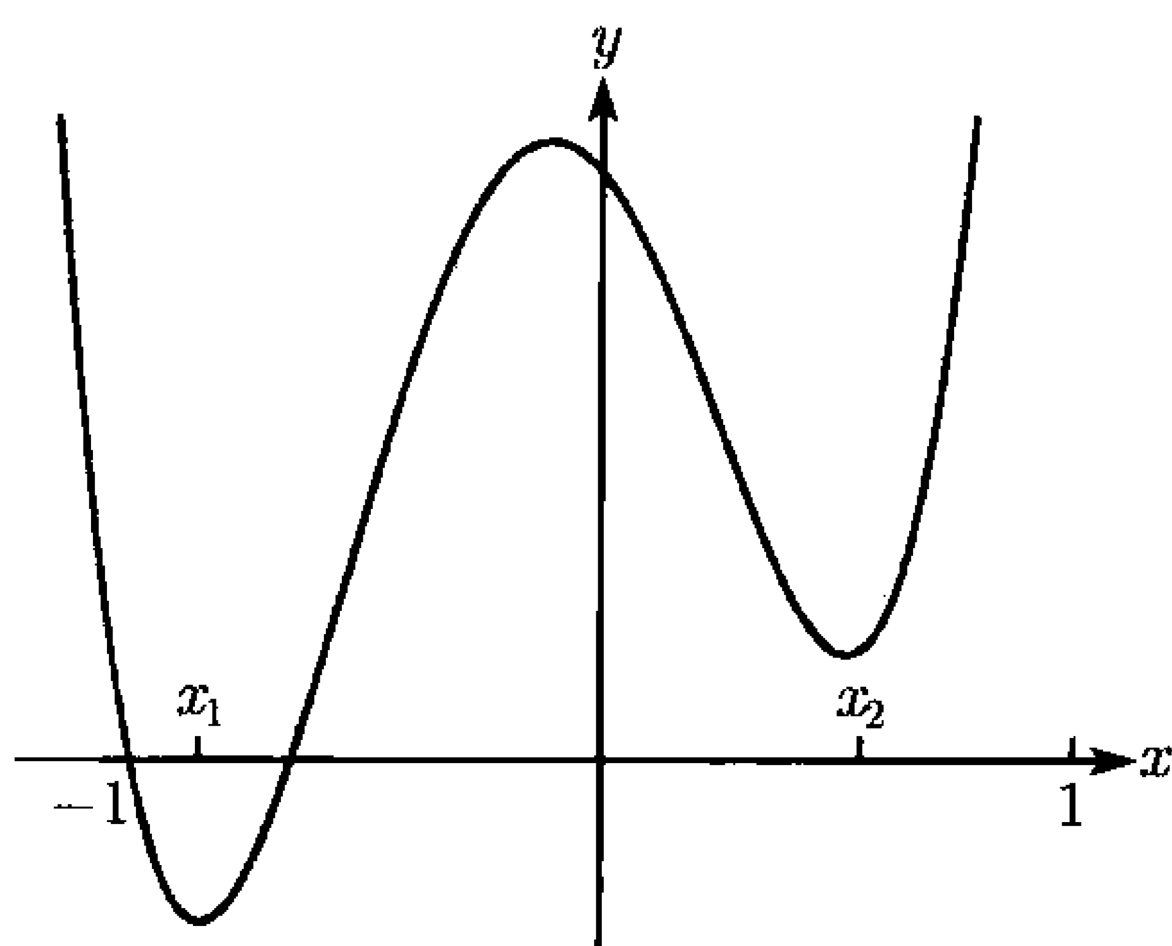


图 13-1　$f(x) = 5x^4 + 3x^3 - 4x^2 - x + 2$ 的最小值问题. 该无约束最小值问题 $\min_x f(x)$ 的解是 x_1

根据是否使用目标函数的导数, 无约束优化的方法分成两组. 如果 $f(x)$ 是代数函数, 在大多数情况下它的导数可以用手算或计算机代数而容易地获得. 如果可

能的话, 应该使用导数的信息, 但是存在一些它可能不行的原因. 特别是目标函数可能太复杂、维数太高或者是不可微的情形下是如此.

13.1 没有导数的无约束最优化

本节假设对每一个给定的 x 都可以得到目标函数 $f(x)$ 的一个值, 但 $f(x)$ 的导数 (或者偏导数, 当 $f(x)$ 是多维时) 不存在. 我们将讨论不使用导数进行优化的 3 种方法: 黄金分割法, 连续抛物线插值法和 Nelder-Mead 法. 前两种方法只能用于一元函数 $f(x)$, 但 Nelder-Mead 可用于多元函数的情形.

13.1.1 黄金分割探索

一旦求解区间已知, 黄金分割法是用于求解一元函数 $f(x)$ 最小值的一种有效方法.

定义 13.1 *称连续函数 $f(x)$ 在区间 $[a,b]$ 上是**单峰的**(unimodal), 如果在 $[a,b]$ 上有且仅有一个相对极小值或极大值且 $f(x)$ 在其他点是严格递减或严格递增.*

一个单峰函数是先单调递增到区间 $[a,b]$ 上的相对极大值处, 然后再单调递减, 或者先单调递减到区间 $[a,b]$ 上的相对极小值处, 再单调递增.

假设 f 是单峰的且在 $[a,b]$ 上有相对极小值. 选择区间内的两点 x_1 和 x_2, 使得 $a < x_1 < x_2 < b$, 图 13-2 是 $[a,b]=[0,1]$ 的情况. 我们将用一个新的、更小的依旧包含相对极小值的区间替代原区间, 按照以下规则: 如果 $f(x_1) \leqslant f(x_2)$, 在下一步中保留区间 $[a,x_2]$; 如果 $f(x_1) > f(x_2)$, 则保留区间 $[x_1,b]$.

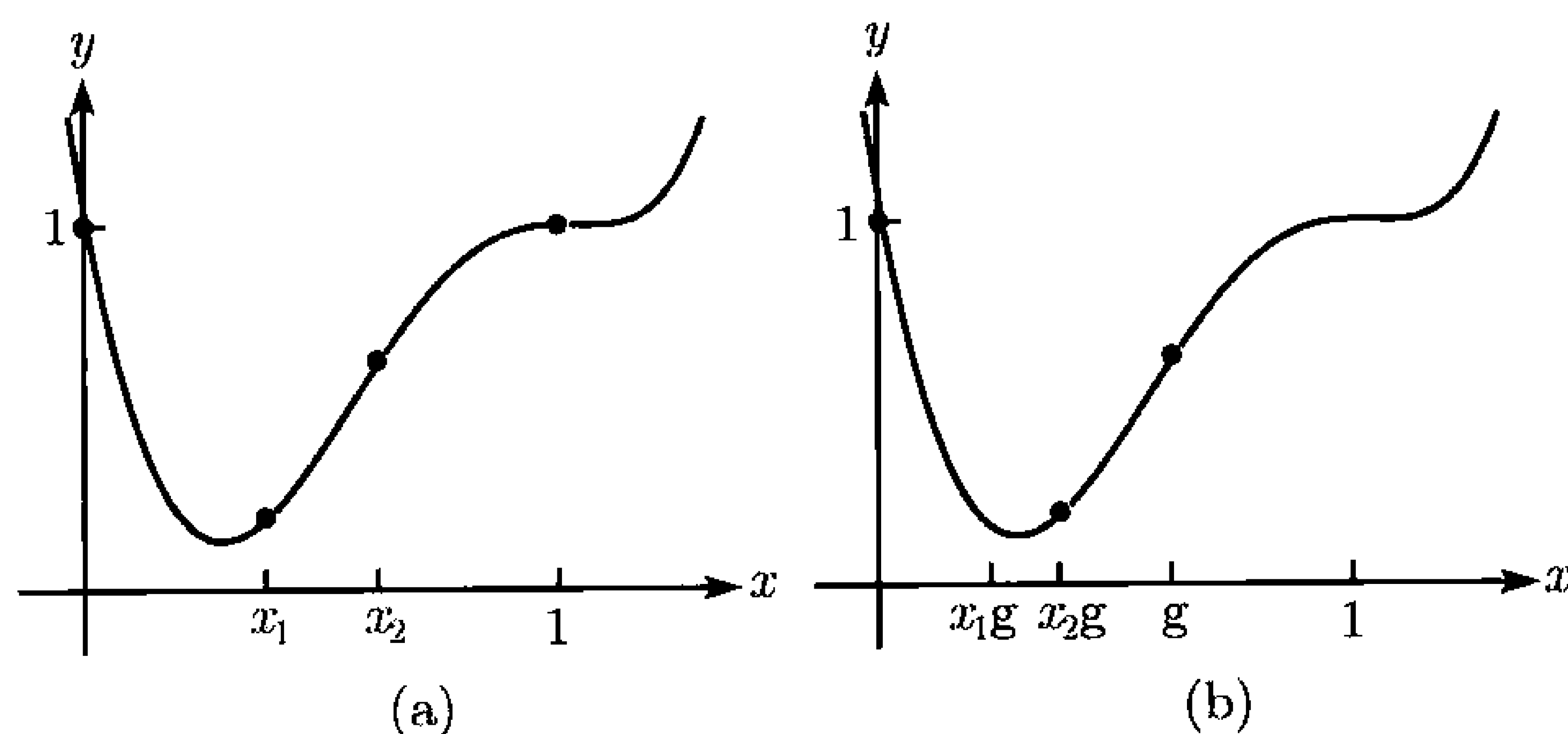


图 13-2 黄金分割法. (a) 在当前区间 $[0,1]$ 上比较在 x_1, x_2 两点的目标函数值. 如果 $f(x_1) < f(x_2)$, 则新区间为 $[0,x_2]$. (b) 在下一步中, 令 $g = x_2$ 且类似地比较 x_1g 和 x_2g

注意到在任意一种情况下, 新的区间包含单峰函数 $f(x)$ 的一个相对极小值. 例如, 如图 13-2 所示, 如果 $f(x_1) < f(x_2)$, 由于单峰假设, 最小值肯定在 x_2 的左边取得. 这是因为 f 一定单调减到最小值的左边, 因此 $f(x_1) < f(x_2)$ 意味着 x_2 一定在最小值的右边. 同样, $f(x_1) > f(x_2)$ 意味着区间 $[x_1,b]$ 包含最小值. 由于新区间比

以前的区间 $[a,b]$ 小, 因此求最小值得以一步步前进. 这样一步步迭代下去, 直到包含最小值的区间足够小. 该方法类似于求根的对分法.

下面讨论在区间 $[a,b]$ 上如何放置 x_1 和 x_2. 在每一步中, 我们想用尽可能少的工作, 尽可能多地减少区间长度. 区间 $[a,b]=[0,1]$ 的上述实现方式如图 13-3 所示. x_1 和 x_2 的选择要满足两个准则: (a) 使得它们关于区间对称 (因为我们没有关于最小值位于区间哪端的任何信息); (b) 选择它们使得不管选哪个作为新区间, x_1 和 x_2 都在下一步中被使用. 也就是说, 要求 (a) $x_1=1-x_2$; (b) $x_1=x_2^2$. 如图 13-3 所示, 如果新区间是 $[0,x_2]$, 准则 (b) 保证原来的 x_1 将是下一个新区间的 x_2; 因此, 只需要计算一次函数值即 $f(x_1g)$ 即可. 同样, 如果新区间是 $[x_1,1]$, 则 x_2 将会成为新的 x_1. 能再次使用函数值表明从第一步之后, 每步只需计算一次目标函数的值就行了.

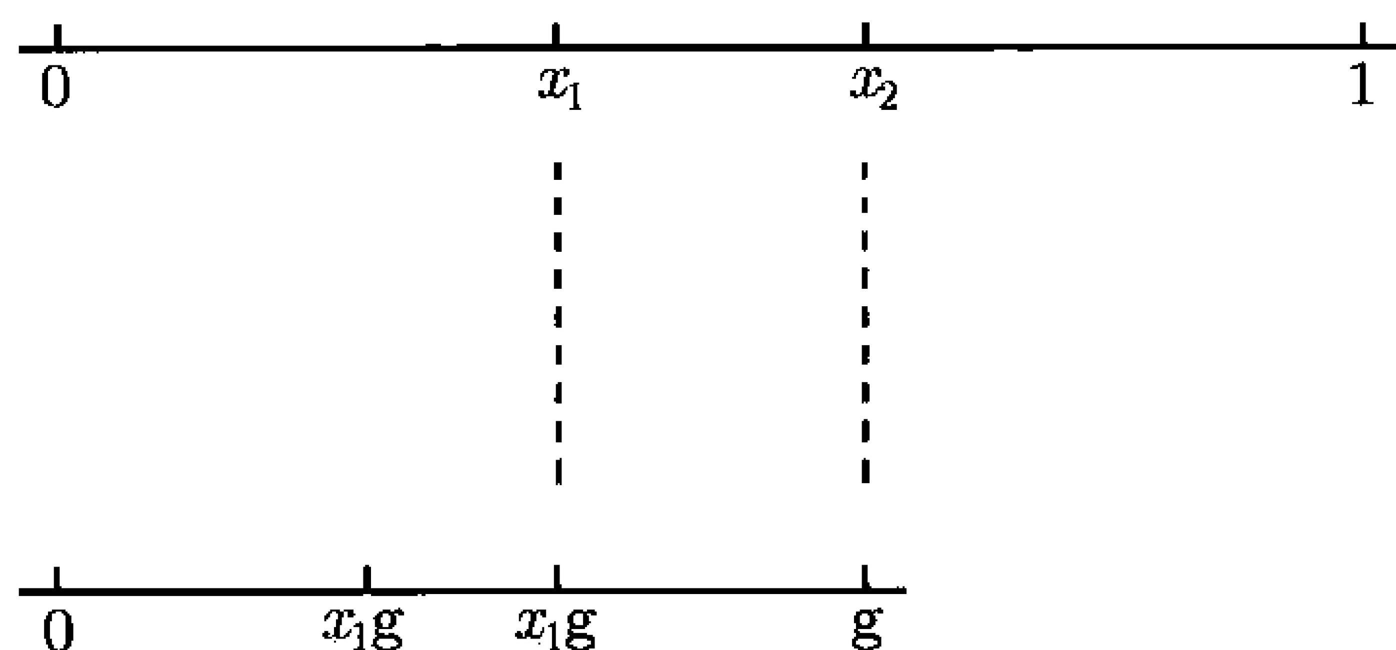


图 13-3 黄金分割法中比例的选取. 上面的线段与下面的线段的比例是 $1/g=(1+\sqrt{5})/2$, 即**黄金分割**. x_1 和 x_2 被精确地选取, 使得不管新区间是 $[0,x_2]$ 还是 $[x_1,1]$, 其中一点被重用为新区间的内点, 从而把每步所需计算目标函数值的次数减少为 1

准则 (a) 和 (b) 合起来意味着 $x_2^2+x_2-1=0$. 这个二次方程的正根是 $x_2=g=(\sqrt{5}-1)/2$. 要使用该算法, 目标函数在 $[a,b]$ 上必须是单峰的且 $f(x)$ 在区间 $[x_1,x_2]$ 内部函数值均存在, 其中

$$a<x_1=a+(1-g)(b-a)<x_2=a+g(b-a)<b.$$

注意到 x_1 和 x_2 是满足 a,b 间比例 $1-g$ 和 g 的值. 按上述方法选择新区间且重复以上步骤. 新区间是原区间长度的 g 倍, 因此在 k 步后当前的区间长度为 $g^k(b-a)$. 最终区间的中点正好是最终区间长度的一半, 即 $g^k(b-a)/2$. 我们已经证明了下述定理:

定理 13.2 *从初始区间 $[a,b]$ 出发, 用黄金分割法 k 步后所得最终区间的中点是最小值的 $g^k(b-a)/2$ 倍, 其中 $g=(\sqrt{5}-1)/2\approx 0.618$.*

黄金分割法

假设 f 是 $[a,b]$ 上有极小值的单峰函数.

for $i=1,2,3,\cdots$

$g=(\sqrt{5}-1)/2$

if $f(a+(1-g)(b-a)) < f(a+g(b-a))$
　　$b = a+g(b-a)$
else
　　$a = a+(1-g)(b-a)$
end
end
最终区间 $[a,b]$ 含有一个最小值.

亮点　收敛性

按照定理 13.2, 黄金分割法以 $g \approx 0.618$ 的速度线性收敛于最小值. 有趣的是该方法和第 1 章中求解方程根的对分法有很多相似点. 虽然它们求解不同的问题, 但两者均全局收敛, 也就是说从合适的条件 (在黄金分割法中函数在 $[a,b]$ 上是单峰的, 在对分法中 $f(a)f(b) < 0$) 出发它们最终都能保证收敛到一个解. 两者均不需要可微. 两者每步均只计算一次函数值且都是线性收敛的. 对分法的收敛速度稍快点, 其线性收敛速度 $K = 0.5 < g = 0.618$. 它们都属于慢而稳的有价值的方法范畴.

黄金分割法的 MATLAB代码要求从第二步起每步计算一个函数值, 这正如前面所说的.

```
% Program 13.1 Golden Section Search for minimum of f(x)
% Start with unimodal f(x) and minimum in [a,b]
% Input: inline function f, interval [a,b], number of steps k
% Output: approximate minimum y
function y=gss(f,a,b,k)
g=(sqrt(5)-1)/2;
x1 = a+(1-g)*(b-a);
x2 = a+g*(b-a);
f1=f(x1);f2=f(x2);
for i=1:k
  if f1 < f2           % if f(x1) < f(x2), replace b with x2
    b=x2; x2=x1; x1=a+(1-g)*(b-a);
    f2=f1; f1=f(x1);   % single function evaluation
  else                 % otherwise, replace a with x1
    a=x1; x1=x2; x2=a+g*(b-a);
    f1=f2; f2=f(x2);   % single function evaluation
  end
end
y=(a+b)/2;
```

例 13.1　用黄金分割法求解函数 $f(x) = x^6 - 11x^3 + 17x^2 - 7x + 1$ 在区间 $[0,1]$ 上的最小值. ◀

图 13-2 展示出了该方法的前两步. 在第一步中, $x_1 = 1-g, x_2 = g$, 其中

$g = (\sqrt{5}-1)/2$. 由于 $f(x_1) < f(x_2)$, 区间 $[0,1]$ 被 $[0,g]$ 取代. 新的 x_1, x_2 分别是以前的 $x_1 g, x_2 g$. 在第二步中, 再次有 $f(x_1) < f(x_2)$, 因此区间 $[0,g]$ 被 $[0,x_2]$ 所替代. 前 15 步的信息如表 13-1 所示.

表 13-1

步数	a	x_1	x_2	b
0	0.000 0	0.382 0	0.618 0	1.000 0
1	0.000 0	0.236 1	0.382 0	0.618 0
2	0.000 0	0.145 9	0.236 1	0.382 0
3	0.145 9	0.236 1	0.291 8	0.382 0
4	0.236 1	0.291 8	0.326 2	0.382 0
5	0.236 1	0.270 5	0.291 8	0.326 2
6	0.270 5	0.291 8	0.305 0	0.326 2
7	0.270 5	0.283 7	0.291 8	0.305 0
8	0.270 5	0.278 6	0.283 7	0.291 8
9	0.278 6	0.283 7	0.286 8	0.291 8
10	0.278 6	0.281 7	0.283 7	0.286 8
11	0.281 7	0.283 7	0.284 9	0.286 8
12	0.281 7	0.282 9	0.283 7	0.284 9
13	0.282 9	0.283 7	0.284 1	0.284 9
14	0.282 9	0.283 4	0.283 7	0.284 1
15	0.283 4	0.283 7	0.283 8	0.284 1

迭代 15 步后, 我们可以说最小值在 0.283 4 和 0.283 8 之间.

13.1.2 连续抛物线插值法

在黄金分割法中, 除了比较 $f(x_1)$ 和 $f(x_2)$ 的大小外, 它们的函数值并没有用到. 不管一个比另一个大多少, 我们都应该决定如何前进. 本节描述一种较少浪费函数值的新方法, 该方法使用函数值建立了函数 f 的局部模型.

选择的局部模型是抛物线, 在第 3 章中我们已经知道它由 3 个点唯一确定. 如图 13-4 所示, 首先在最小值附近选 3 个点 r, s, t. 计算在这 3 点的目标函数 f 的值且画出过这 3 点的抛物线. 差分如下:

$$
\begin{array}{c|ccc}
r & f(r) & & \\
 & & d_1 & \\
s & f(s) & & d_3 \\
 & & d_2 & \\
t & f(t) & &
\end{array}
$$

其中 $d_1 = (f(s)-f(r))/(s-r)$, $d_2 = (f(t)-f(s))/(t-s)$, $d_3 = (d_2-d_1)/(t-r)$. 因

此可得抛物线如下:

$$P(x) = f(r) + d_1(x-r) + d_3(x-r)(x-s). \tag{13.1}$$

令 $P(x)$ 的导数为 0 去求抛物线的最小值, 得到的公式

$$x = \frac{r+s}{2} - \frac{(f(s)-f(r))(t-r)(t-s)}{2[(s-r)(f(t)-f(s))-(f(s)-f(r))(t-s)]} \tag{13.2}$$

是最小值的新的近似. 在连续抛物线算法中新的 x 将替代 r, s, t 中最远的或最次 (least optimal) 的点, 如果需要的话则重复该过程. 不像黄金分割法那样, 连续抛物线算法不一定收敛. 但如果它确实收敛的话则速度要更快一些, 因为它更明智地使用了函数值的信息.

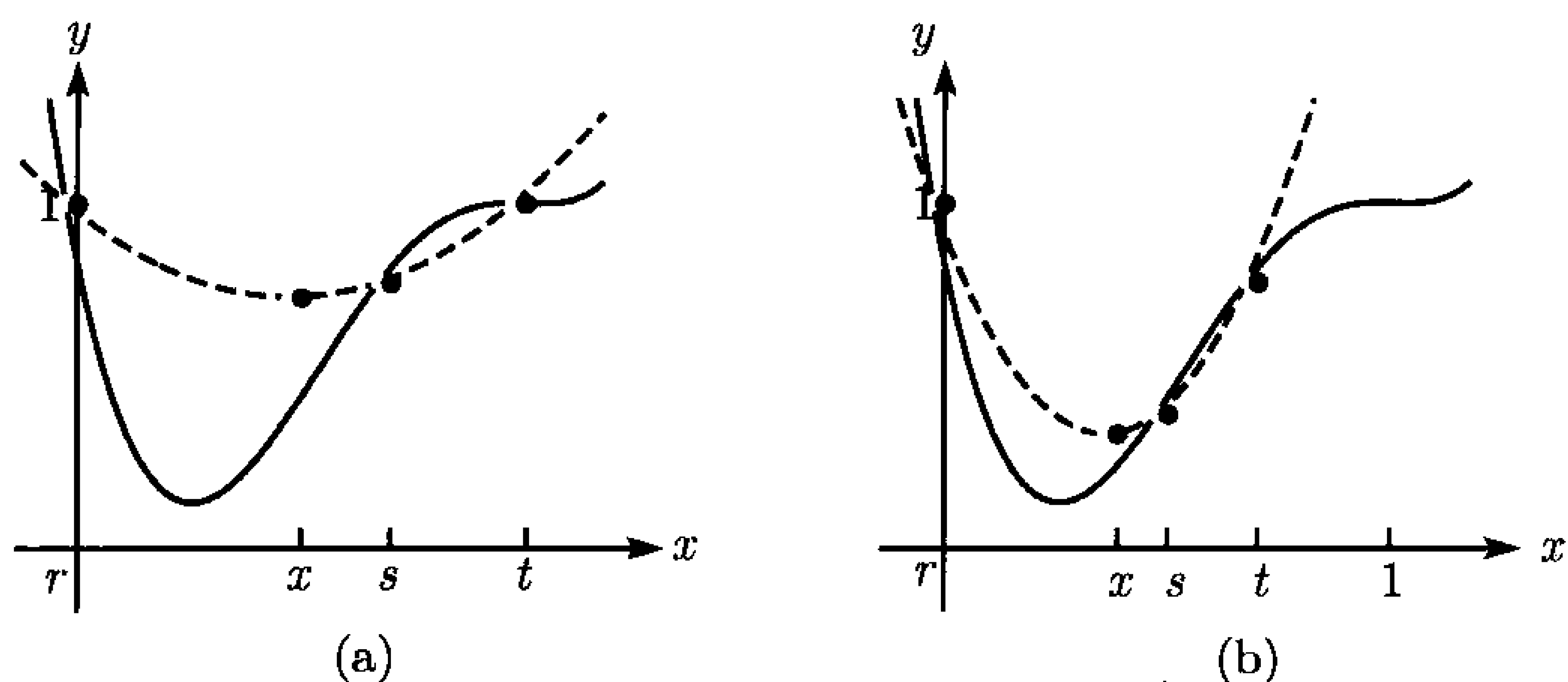


图 13-4　连续抛物线插值算法. (a) 过当前 3 点 r, s, t 的抛物线且抛物线的最小值 x 用于代替当前点 s. (b) 用新的 r, s, t 重复该步骤

连续抛物线插值算法

从极小值点 r, s, t 开始

for $i = 1, 2, 3, \cdots$

$$x = \frac{r+s}{2} - \frac{(f(s)-f(r))(t-r)(t-s)}{2[(s-r)(f(t)-f(s))-(f(s)-f(r))(t-s)]}$$

$t = s$

$s = r$

$r = x$

end

在下面的 MATLAB代码中, 抛物线的最小值代替了当前 3 个点中的最远点:

```
% Program 13.2 Successive Parabolic Interpolation
% Input: inline function f, initial guesses r,s,t, steps k
% Output: approximate minimum x
function x=spi(f,r,s,t,k)
x(1)=r;x(2)=s;x(3)=t;
fr=f(r);fs=f(s);ft=f(t);
```

```
for i=4:k+3
 x(i)=(r+s)/2-(fs-fr)*(t-r)*(t-s)/(2*((s-r)*(ft-fs)
      -(fs-fr)*(t-s)));
 t=s;s=r;r=x(i);
 ft=fs;fs=fr;fr=f(r);          % single function evaluation
end
```

例 13.2 用连续抛物线算法求 $f(x)=x^6-11x^3+17x^2-7x+1$ 在区间 $[0,1]$ 上的最小值. ◀

选用初始点 $r=0, s=0.7, t=1$, 我们计算如表 13-2 所示的几步.

表 13-2

步数	x	$f(x)$
0	1.000 000 000 000 00	1.000 000 000 000 00
0	0.700 000 000 000 00	0.774 649 000 000 00
0	0.000 000 000 000 00	1.000 000 000 000 00
1	0.500 000 000 000 00	0.390 625 000 000 00
2	0.385 896 835 485 38	0.201 472 878 145 00
3	0.331 751 296 025 24	0.148 441 657 246 73
4	0.237 355 733 167 21	0.149 337 377 644 02
5	0.285 266 172 693 72	0.131 726 603 381 64
6	0.285 169 421 616 39	0.131 724 261 362 34
7	0.283 740 694 642 18	0.131 706 464 517 92
8	0.283 646 476 311 23	0.131 706 398 590 35
9	0.283 648 264 375 69	0.131 706 398 563 01
10	0.283 648 358 329 62	0.131 706 398 562 95
11	0.283 648 358 083 77	0.131 706 398 562 95
12	0.283 648 332 187 29	0.131 706 398 562 95

我们推断出最小值在 $x_{\min}=0.283\,648\,3$ 附近. 注意到 12 步之后我们已经远远胜过了使用更少函数值的黄金分割法. 尽管使用了函数 f 的精确值, 但我们并没有使用目标函数的导数信息, 然而黄金分割法仅仅用到了函数值的比较.

从表的结尾处我们产生了好奇心. 像在第 1 章首先讨论的那样, 函数在相对极大值和极小值附近趋于平坦. 由于 $x_{\min}$ 精确到 10^{-7} 有相同的极小函数值, 因此当使用 IEEE 双精度时不管迭代多少步我们都不可能超出这个精确值. 由于最小值一般出现在函数导数为零的点处, 因此这个困难不是优化方法的失误, 而是浮点计算的局限性.

从 GSS 到 SPI 的进步类似于从对分法到割线法以及逆二次插值. 建立函数的一个局部模型而且运作起来似乎是目标函数有助于加速收敛.

13.1.3 Nelder-Mead 搜索

对一个多元函数来说, 方法变得更复杂. Nelder-Mead 搜索试图将一个多面体"下山"式地展开到最低可能的层次. 因此, 它又称为**下山单纯形法**(downhill simplex method). 它不使用目标函数的导数.

假设要极小化有 n 个自变量的函数 f. 该方法用 $n+1$ 个 n 维空间中的猜测向量 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{n+1}$ 作为初始向量, 它们构成了 n 维单纯形的顶点. 例如, 如果 $n=2$, 则初始的 3 个猜测向量形成了平面中三角形的顶点.

单纯形中的顶点被检验且根据它们函数值 $y_1 < y_2 < \cdots < y_{n+1} = y_h$ 的大小升序排列. 最不优的单纯形向量 $\boldsymbol{x}_h = \boldsymbol{x}_{n+1}$ 将按照图 13-5 的流程图被替代. 首先记除去 $\boldsymbol{x}_h$ 的单纯形表面的质心为 $\overline{\boldsymbol{x}}$. 然后像图 13-5a 中所示, 我们检查关于反射点 $\boldsymbol{x}_r = 2\overline{\boldsymbol{x}} - \boldsymbol{x}_h$ 的函数值 $y_r = f(\boldsymbol{x}_r)$. 如果新的 y_r 的值满足 $y_1 < y_r < y_n$, 我们将用 $\boldsymbol{x}_r$ 替代最坏的点 $\boldsymbol{x}_n$, 按函数值将顶点排序, 且重复这个步骤.

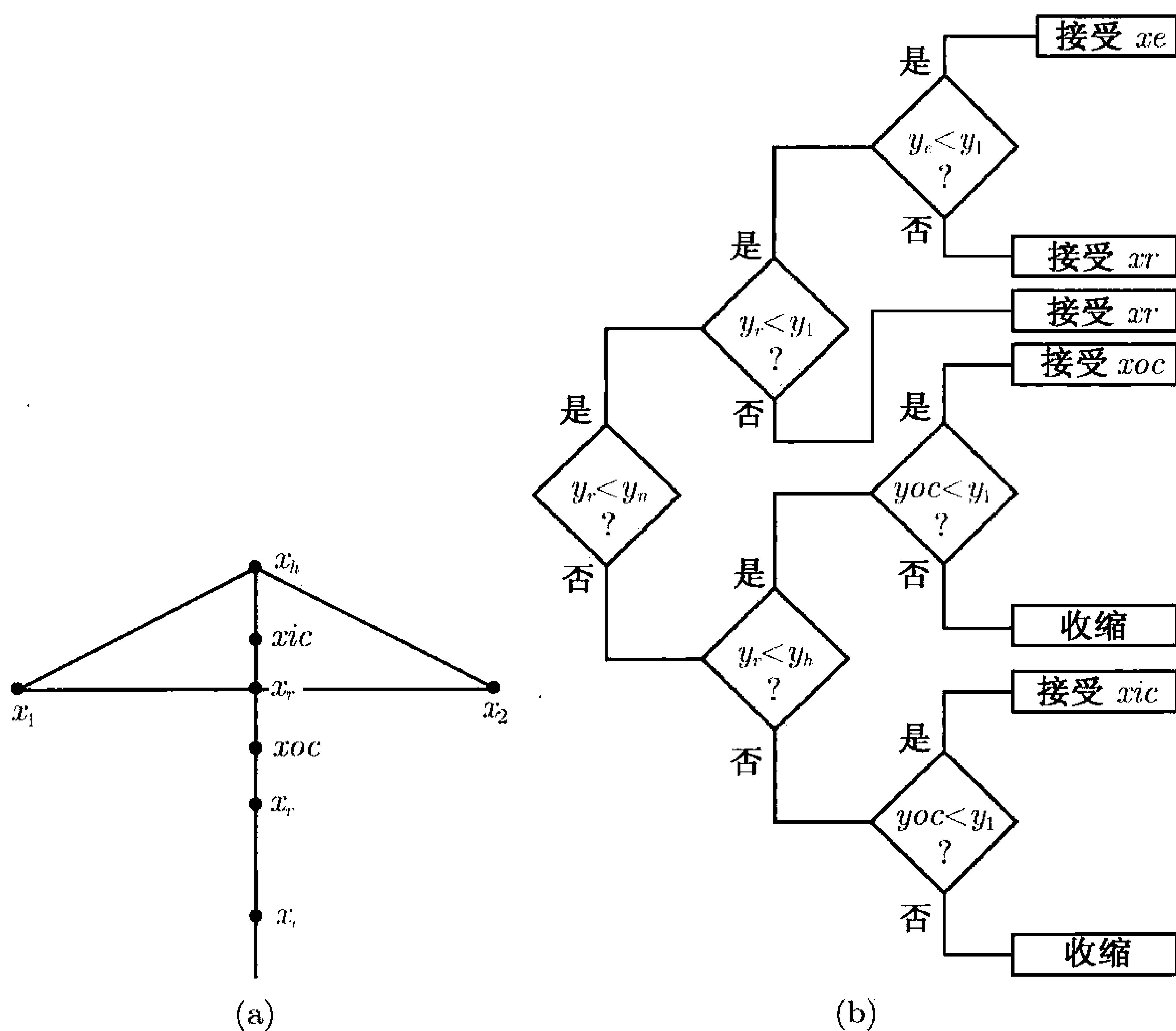


图 13-5 Nelder-Mead 搜索. (a) 最大函数值点 x_h 和质心点 $\overline{x}$ 连线上的点被检测. (b) 描述该方法一步的流程图

如果 y_r 比当前的最小值 y_1 小, 那么设法进行外推, 用 $\boldsymbol{x}_e = 3\overline{\boldsymbol{x}} - 2\boldsymbol{x}_h$ 来看我们是否在该方向应前进更多. 对这个步骤来说更好的 $\boldsymbol{x}_e$ 和 $\boldsymbol{x}_r$ 是被接受的. 另一方面, 如果 y_r 比 y_n(当前的最大值, 一旦 x_{n+1} 被忽略的话) 大, 那么不论是在如图

所示的外收缩点 $\boldsymbol{x}_{oc}=1.5\overline{\boldsymbol{x}}-0.5\boldsymbol{x}_h$ 还是在内收缩点 $\boldsymbol{x}_{ic}=0.5\overline{\boldsymbol{x}}+0.5\boldsymbol{x}_h$, 都要进一步检测. 在这两个点中任何一个不能改进意味着通过扩充并不能前进, 该算法应寻求更靠近最优值的点. 在进行下一步之前, 先在当前最小值 x_1 的方向缩小到原来的 $\frac{1}{2}$. MATLAB代码如下, 内联函数 f 是关于变量 x(1),x(2),...,x(n) 的.

```
% Program 13.3 Nelder-Mead Search
% Input: inline function f, best guess xbar (column vector),
%    initial search radius rad and number of steps k
% Output: matrix x whose columns are vertices of simplex,
%    function values y of those vertices
function [x,y]=neldermead(f,xbar,rad,k)
n=length(xbar);
x(:,1)=xbar;           % each column of x is a simplex vertex
x(:,2:n+1)=xbar*ones(1,n)+rad*eye(n,n);
for j=1:n+1
  y(j)=f(x(:,j));      % evaluate obj function f at each vertex
end
[y,r]=sort(y);         % sort the function values in ascending order
x=x(:,r);              %    and rank the vertices the same way
for i=1:k
  xbar=mean(x(:,1:n)')';  % xbar is the centroid of the face
  xh=x(:,n+1);            %   omitting the worst vertex xh
  xr = 2*xbar - xh; yr = f(xr);
  if yr < y(n)
    if yr < y(1)       % try expansion xe
      xe = 3*xbar - 2*xh; ye = f(xe);
      if ye < yr       % accept expansion
        x(:,n+1) = xe; y(n+1) = f(xe);
      else             % accept reflection
        x(:,n+1) = xr; y(n+1) = f(xr);
      end
    else               % xr is middle of pack, accept reflection
      x(:,n+1) = xr; y(n+1) = f(xr);
    end
  else                 % xr is still the worst vertex, contract
    if yr < y(n+1)     % try outside contraction xoc
      xoc = 1.5*xbar - 0.5*xh; yoc = f(xoc);
      if yoc < yr      % accept outside contraction
        x(:,n+1) = xoc; y(n+1) = f(xoc);
      else             % shrink simplex toward best point
        for j=2:n+1
          x(:,j) = 0.5*x(:,1)+0.5*x(:,j); y(j) = f(x(:,j));
        end
      end
    else               % xr is even worse than the previous worst
      xic = 0.5*xbar+0.5*xh; yic = f(xic);
      if yic < y(n+1) % accept inside contraction
        x(:,n+1) = xic; y(n+1) = f(xic);
```

```
      else              % shrink simplex toward best point
        for j=2:n+1
          x(:,j) = 0.5*x(:,1)+0.5*x(:,j); y(j) = f(x(:,j));
        end
      end
    end
  end
  [y,r] = sort(y);      % resort the obj function values
  x=x(:,r);             %    and rank the vertices the same way
end
```

代码实现的流程图如图 13-5b 所示. 迭代步数需要输入. 计算机问题 8 要求读者根据用户给定的容许误差界的停止准则重新写该程序. 一般的停止准则既要求单纯形大小的减少在容许的范围内, 又要求在顶点处函数最大值的传播在小的容许范围内. MATLAB用它的 `fminsearch` 命令实现 Nelder-Mead 方法.

习题 13.1

1. 证明下面函数在某区间上是单峰的, 并且找出它的绝对极小值和取到该极小值的点.
   (a) $f(x) = \mathrm{e}^x + \mathrm{e}^{-x}$; (b) $f(x) = x^6$;
   (c) $f(x) = 2x^4 + x$; (d) $f(x) = x - \ln x$.
2. 找出给定区间上的绝对极小值和取到该极小值的点.
   (a) $f(x) = \cos x, x \in [3,4]$; (b) $f(x) = 2x^3 + 3x^2 - 12x + 3, x \in [0,2]$;
   (c) $f(x) = x^3 + 6x^2 + 5, x \in [-5,5]$; (d) $f(x) = 2x + \mathrm{e}^{-x}, x \in [-5,5]$.

计算机问题 13.1

1. 画出函数 $f(x)$ 的图像, 找出长度唯一的初始区间, 在该区间上 $f(x)$ 在相对极小值附近是单峰的. 然后用 GSS 去查找函数的每一个相对极小值, 保留 5 位有效数字.
   (a) $f(x) = 2x^4 + 3x^2 - 4x + 5$; (b) $f(x) = 3x^4 + 4x^3 - 12x^2 + 5$;
   (c) $f(x) = x^6 + 3x^4 - 2x^3 + x^2 - x - 7$; (d) $f(x) = x^6 + 3x^4 - 12x^3 + x^2 - x - 7$.
2. 对计算机问题 1 中的函数用 SPI. 查找函数的极小点, 保留 5 位有效数字.
3. 用两种不同的方法去找双曲线 $y = 1/x$ 上离点 (2,3) 最近的点: (a) 用牛顿法找临界点; (b) 关于锥上的点和点 (2,3) 间的距离的平方, 用黄金分割法求解.
4. 用计算机问题 3 中 (a), (b) 的方法, 求解椭圆 $4x^2 + 9y^2 = 4$ 上离点 (1,5) 最远的点.
5. 用 Nelder-Mead 方法求解函数 $f(x,y) = \mathrm{e}^{-x^2y^2} + (x-1)^2 + (y-1)^2$ 的最小值. 试着用不同的初始条件, 并且比较结果. 用该方法, 你能精确到几位有效数字?
6. 用 Nelder-Mead 方法求解下列函数的最小值, 精确到小数点后 6 位 (每个函数有两个最小值):
   (a) $f(x,y) = x^4 + y^4 + 2x^2y^2 + 6xy - 4x - 4y + 1$
   (b) $f(x,y) = x^6 + y^6 + 3x^2y^2 - x^2 - y^2 - 2xy$
7. 用 Nelder-Mead 方法求解 Rosenbrock 函数 $f(x,y) = 100(y - x^2)^2 + (x-1)^2$ 的最小值.
8. 重写程序 13.3, 使停止准则与基于用户指定的误差容限的 Nelder-Mead 方法相适应. 通过对计算机问题 6 中的目标函数求极小值, 精确到小数点后 6 位, 来说明这一点.

13.2 带导数的无约束最优化

导数含有关于函数值增加和减少的速度的信息, 偏导数还表明最快增加和减少的方向. 如果关于目标函数的导数信息可用, 那么可以使用它来更有效地去求最优解.

13.2.1 牛顿法

如果函数是连续可微的且可以计算导数, 那么最优化问题可以表示成一个求根问题. 我们首先考虑一维的情况, 该情况的解释是最简单的.

一个连续可微的函数 $f(x)$ 在最小值 x^* 处的一阶导数一定为零. 第 1 章中的方法可用于求解新方程 $f'(x)=0$ 的解. 如果目标函数是单峰的且在区间上有最小值, 则以最小值 x^* 的附近点作为初始点, 用牛顿法将会收敛到 x^*. 用于 $f'(x)=0$ 的牛顿法形成迭代

$$x_{k+1}=x_k-\frac{f'(x_k)}{f''(x_k)}. \tag{13.3}$$

然而牛顿法 (13.3) 将找到 $f'(x)=0$ 的点, 一般来说, 这些点不一定是最小值点. 找到最优值的一个合理的初始估计且一旦找到后检查这些点的最优性是很重要的.

用这种方法优化函数 $f(x_1,\cdots,x_n)$ 会用到多元函数的牛顿法. 正如在一维情形一样, 我们想要让导数为零然后求解. 因此, 我们得到

$$\nabla f=0, \tag{13.4}$$

其中

$$\nabla f=\left[\frac{\partial f}{\partial x_1}(x_1,\cdots,x_n),\cdots,\frac{\partial f}{\partial x_n}(x_1,\cdots,x_n)\right]$$

表示 f 的梯度.

第 2 章中向量值函数的牛顿法保证 (13.4) 可求解. 令 $F(x)=\nabla f(x)$, 牛顿法的迭代步为 $x_{k+1}=x_k+v$, 其中 v 是 $\mathrm{D}F(x_k)v=-F(x_k)$ 的解. 梯度的雅可比矩阵 $\mathrm{D}F$ 是

$$H_f=\mathrm{D}F=\begin{bmatrix}\dfrac{\partial^2 f}{\partial x_1\partial x_1} & \cdots & \dfrac{\partial^2 f}{\partial x_1\partial x_n}\\ \vdots & & \vdots\\ \dfrac{\partial^2 f}{\partial x_n\partial x_1} & \cdots & \dfrac{\partial^2 f}{\partial x_n\partial x_n}\end{bmatrix}, \tag{13.5}$$

这是 f 的 Hessian 矩阵. 因此牛顿法变为

$$\begin{cases} H_f(x_k)v = -\nabla f(x_k) \\ x_{k+1} = x_k + v \end{cases} \tag{13.6}$$

例 13.3 用牛顿法求解函数 $f(x,y) = 5x^4 + 4x^2y - xy^3 + 4y^4 - x$ 的最小值. ◀

函数如图 13-6 所示. 它的梯度为 $\nabla f = (20x^3 + 8xy - y^3 - 1, 4x^2 - 3xy^2 + 16y^3)$, Hessian 矩阵为

$$H_f(x,y) = \begin{bmatrix} 60x^2 + 8y & 8x - 3y^2 \\ 8x - 3y^2 & -6xy + 48y^2 \end{bmatrix}.$$

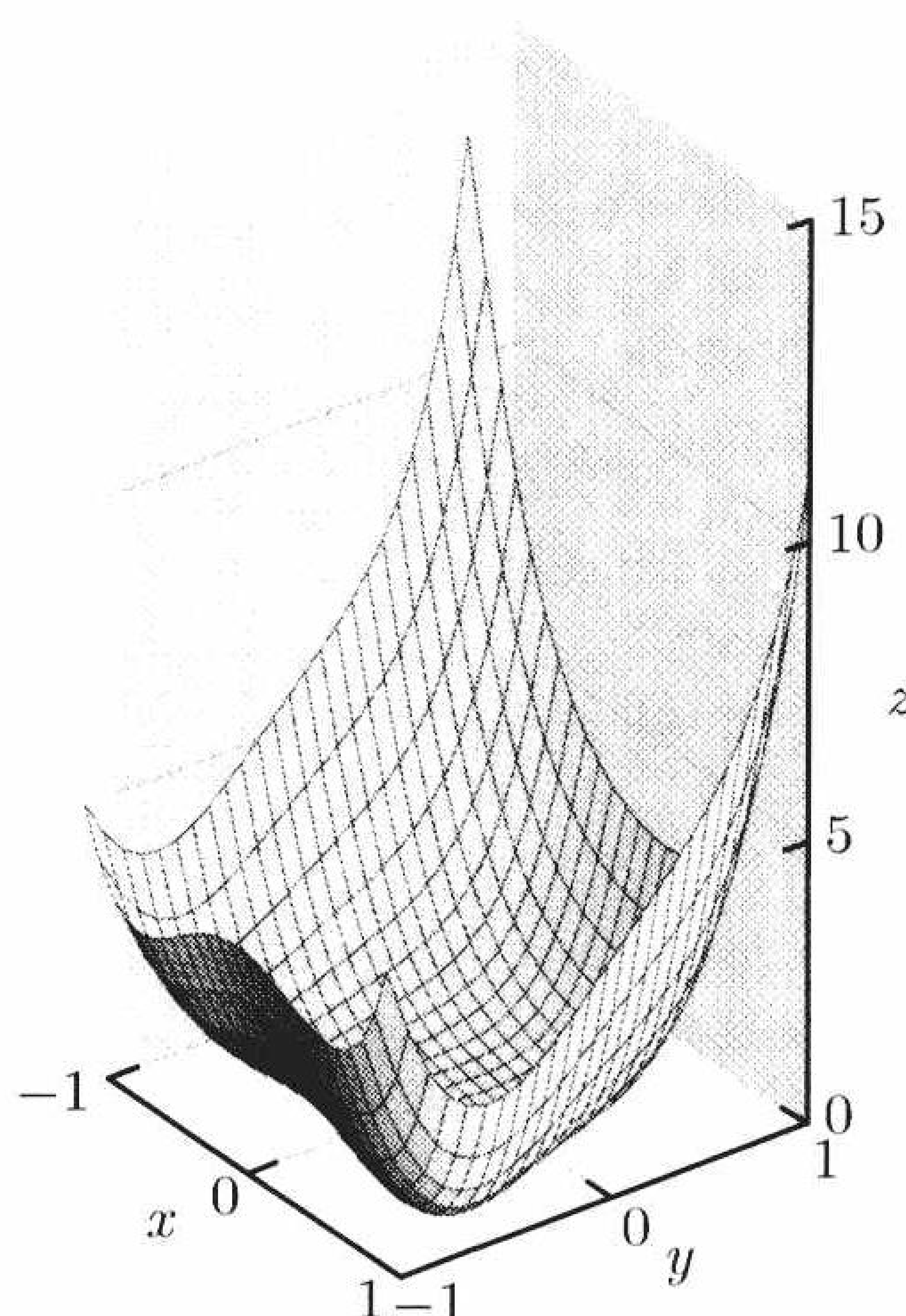


图 13-6 二维函数的表面图: $z = 5x^4 + 4x^2y - xy^3 + 4y^4 - x$ 的图. 用牛顿法求得最小值大约出现在 $(0.492\,3, -0.364\,3)$

用牛顿法 (13.6) 迭代 10 次, 得到如表 13-3 所示的结果.

表 13-3

步数	x	y	$f(x,y)$
0	1.000 000 000 000 00	1.000 000 000 000 00	11.000 000 000 000 00
1	0.644 295 302 013 42	0.637 583 892 617 45	1.770 018 678 274 22
2	0.430 640 345 429 56	0.392 332 987 022 31	0.101 120 065 375 34
3	0.338 779 714 333 52	0.198 577 141 607 17	−0.178 185 859 772 25
4	0.500 097 336 967 80	−0.447 719 295 197 63	−0.429 640 650 539 18
5	0.497 373 505 714 30	−0.379 726 457 286 44	−0.456 737 196 647 08
6	0.492 550 006 518 77	−0.364 977 537 465 14	−0.457 520 090 077 57

(续)

步数	x	y	$f(x,y)$
7	0.492 308 317 591 06	−0.364 287 045 691 73	−0.457 521 622 627 01
8	0.492 307 786 726 81	−0.364 285 559 933 21	−0.457 521 622 634 07
9	0.492 307 786 724 34	−0.364 285 559 926 34	−0.457 521 622 634 07
10	0.492 307 786 724 34	−0.364 285 559 926 34	−0.457 521 622 634 07

牛顿法在计算机精度范围内收敛到最小值 −0.457 5 附近. 注意用牛顿法求解最小化问题的另一个特点: 在解中我们获得了机器精度, 不像一维的 SPI 那样. 原因是我们不再作用于目标函数而是把问题重新转化为涉及梯度的求根问题. 由于 ∇f 在最优值处有单根, 因此让前向误差向机器误差靠近是不困难的.

如果可以计算函数的 Hessian 矩阵, 我们一般选牛顿法. 在二维的情况下, Hessian 矩阵通常是存在的. 在 n 维情况下, 仅仅可以计算 n 维向量在每一个点的梯度, 但却无法构造 $n\times n$ 的 Hessian 阵. 下面的两种方法通常比牛顿法要收敛的慢, 但仅仅需要计算在各个点的梯度值.

13.2.2 最速下降法

最速下降法(steepest descent)又称**梯度法**(gradient search), 它的基本思想是从当前点沿着下降最快的方向来搜索函数的最小值. 由于梯度 ∇f 指向 f 增加最快的方向, 因此它的反方向 $-\nabla f$ 是最速下降方向. 我们应该沿着这个方向走多长呢? 现在把问题降阶为沿直线的极小化问题, 用一维方法确定该走多远. 在沿着最速下降方向找到新的最小值之后, 从这个点开始, 重复这个过程. 也就是说, 找到在新的点处的梯度, 且在新的方向作一维最小化.

最速下降法是一个迭代循环.

最速下降法

for $i=0,1,2,\cdots$

  $v=\nabla f(x_i)$

  对于标量的 $s=s^*$ 极小化 $f(x-sv)$

  $x_{i+1}=x_i-s^*v$

end

我们将最速下降法用于例 13.3 的目标函数.

例 13.4 用最速下降法求解函数 $f(x,y)=5x^4+4x^2y-xy^3+4y^4-x$ 的最小值.

按前述的步骤, 我们用 SPI 求解一维最小化问题. 25 步的迭代结果如表 13-4.

表 13-4

步数	x	y	$f(x,y)$
0	1.000 000 000 000 00	−1.000 000 000 000 00	11.000 000 000 000 00
5	0.403 145 795 181 13	−0.279 920 882 717 56	−0.419 648 888 306 51
10	0.491 968 950 851 12	−0.362 164 043 742 06	−0.457 506 805 237 54
15	0.492 282 844 337 76	−0.364 266 356 861 72	−0.457 521 619 340 16
20	0.492 307 864 175 32	−0.364 285 395 672 77	−0.457 521 622 633 89
25	0.492 307 782 621 42	−0.364 285 565 780 33	−0.457 521 622 634 07

毫无疑问, 与牛顿法相比它的收敛速度慢. 这是因为牛顿法是求解一个方程且使用一阶和二阶导数信息 (包括 Hessian 矩阵), 而最速下降法实际上是沿着下降方向最小化且仅仅使用一阶导数信息. ◀

13.2.3 共轭梯度法

在第 2 章中, 共轭梯度法用于求解对称正定矩阵方程. 现在从不同的方向回到这种方法上来.

当 $\boldsymbol{A}$ 是对称正定矩阵时, 求解 $\boldsymbol{Ax}=\boldsymbol{b}$ 等价于求解一个抛物面的最小值. 例如, 在二维的情形下, 线性系统

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}=\begin{bmatrix} e \\ f \end{bmatrix} \tag{13.7}$$

的解是抛物面

$$f(x_1,x_2)=\frac{1}{2}ax_1^2+bx_1x_2+\frac{1}{2}cx_2^2-ex_1-fx_2 \tag{13.8}$$

的最小值. 原因是 f 的梯度为

$$\nabla f=[ax_1+bx_2-e,bx_1+cx_2-f].$$

在最小值处的梯度为零, 这就得到了前面的矩阵方程. 正定意味着抛物面是上凹的.

关键的观察是, 线性系统 (13.7) 的残差 $\boldsymbol{r}=\boldsymbol{b}-\boldsymbol{Ax}$ 是 $-\nabla f(\boldsymbol{x})$, 即函数 f 在 $\boldsymbol{x}$ 点处的最速下降方向. 假设已经选了一个搜索方向, 记为向量 $\boldsymbol{d}$. 沿着那个方向极小化 (13.8) 中的 f, 等价于求使得函数 $h(\alpha)=f(\boldsymbol{x}+\alpha\boldsymbol{d})$ 最小的 α. 令导数为零得到最小值:

$$0=\nabla f\cdot\boldsymbol{d}=(\boldsymbol{A}(\boldsymbol{x}+\alpha\boldsymbol{d})-(e,f)^{\mathrm{T}})\cdot\boldsymbol{d}=(\alpha\boldsymbol{Ad}-\boldsymbol{r})^{\mathrm{T}}\boldsymbol{d}.$$

这意味着

$$\alpha=\frac{\boldsymbol{r}^{\mathrm{T}}\boldsymbol{d}}{\boldsymbol{d}^{\mathrm{T}}\boldsymbol{Ad}}=\frac{\boldsymbol{r}^{\mathrm{T}}\boldsymbol{r}}{\boldsymbol{d}^{\mathrm{T}}\boldsymbol{Ad}},$$

其中最后一个等式是由关于共轭梯度方法的定理 2.13 得到的.

由上面的计算, 我们推断可以选用共轭梯度法求解一个抛物面的最小值, 但是要代之以

$$r_i = -\nabla f$$

和

$$\alpha_i = 使\ f(x_{i-1} + \alpha d_{i-1})\ 极小化的\ \alpha$$

事实上, 从这种方式看, 注意到我们已经完全用 f 表示了共轭梯度. 不再涉及 $\boldsymbol{A}$. 对一般的函数 f, 可以用这种形式的方法. 在 f 有抛物线形状的区域附近, 这种方法将快速地移向底部. 新的算法有以下步骤:

共轭梯度法

令 x_0 为初始猜测, 并设 $d_0 = r_0 = -\nabla f$

for 使 $f(x_{i-1} + \alpha d_{i-1})$ 最小化的 $i = 1, 2, 3, \cdots$

$$x_i = x_{i-1} + \alpha_i d_{i-1}$$

$$r_i = -\nabla f(x_i)$$

$$\beta_i = \frac{r_i^{\mathrm{T}} r_i}{r_{i-1}^{\mathrm{T}} r_{i-1}}$$

$$d_i = r_i + \beta_i d_{i-1}$$

end

我们将该新方法用于一个熟悉的例子.

例 13.5 用共轭梯度法, 求解函数 $f(x,y) = 5x^4 + 4x^2y - xy^3 + 4y^4 - x$ 的最小值.

按前述的步骤, 用 SPI 求解一维最小化问题. 20 步的迭代结果如表 13-5.

表 13-5

步数	x	y	$f(x,y)$
0	1.000 000 000 000 00	1.000 000 000 000 00	11.000 000 000 000 00
5	0.460 386 575 999 35	−0.383 161 140 298 60	−0.448 499 534 206 21
10	0.490 488 928 071 81	−0.361 065 611 278 30	−0.457 484 771 714 84
15	0.492 437 149 561 28	−0.364 216 614 735 26	−0.457 521 476 043 12
20	0.492 314 777 515 83	−0.364 298 172 753 71	−0.457 521 622 069 84

无约束优化的范围很广, 这一章的方法仅仅代表冰山一角. 像 SPI 或共轭梯度法那样, **信赖域方法**(trust region method)形成了局部模型, 但是它们仅仅适用于一特定的区域, 该区域在搜索前进时变窄. MATLAB 优化工具箱中的期 `fminunc` 程序就是信赖域方法的一个例子. **模拟退火**(simulated annealing) 是一种随机算法, 它试图使目标函数前进较慢, 但会小的正的概率接受向上步, 这是为了避免收敛到非

最优的局部极小值. 一般而言, **遗传算法**(genetic algorithm)和进化计算提出了全新的优化方法并且人们现在仍然在积极探究它们.

约束优化是在一系列约束下求一个目标函数的最小值. 这些问题中最常见的是线性规划, 自从 20 世纪中期形成后它已经可以通过单纯形方法求解, 尽管最近基于内点法的新的更快的算法已经出现. 二次非线性规划问题需要更复杂的方法. 参考有关这方面的文献. ◀

计算机问题 13.2

1. 用牛顿法求函数 $f(x,y)=\mathrm{e}^{-x^2y^2}+(x-1)^2+(y-1)^2$ 的最小值. 取不同的初始条件, 比较所得的结果. 用这种方法你能得到几位有效数字?
2. 用牛顿法求下面函数的最小值, 精确到小数点后 6 位 (每个函数有两个最小值).
   (a) $f(x,y)=x^4+y^4+2x^2y^2+6xy-4x-4y+1$;
   (b) $f(x,y)=x^6+y^6+3x^2y^2-x^2-y^2-2xy$.
3. 求 Rosenbrock 函数 $f(x,y)=100(y-x^2)^2+(x-1)^2$ 的最小值, 通过使用 (a) 牛顿法; (b) 最速下降法. 用初始点 $(2,2)$. 经过多少步之后可得到解? 解释一下它们所获得的精度的差异.
4. 用最速下降法求计算机问题 2 中函数的最小值.
5. 用共轭梯度法求计算机问题 2 中函数的最小值.
6. 用共轭梯度法求下列函数的最小值, 保留 5 位有效数字.
   (a) $f(x,y)=x^4+2y^4+3x^2y^2+6x^2y-3xy^2+4x-2y$;
   (b) $f(x,y)=x^6+x^2y^4+y^6+3x+2y$.

实例检验 13 分子结构和数值优化

蛋白质的功能和它形式有关: 蛋白质的形状决定了它的功能, 分子形状的节和折痕使它们捆绑和成块——这是它们一起作用所必需的. 控制氨基酸构成 (折叠) 为蛋白质的力是由于原子间的化学键和自由原子间更弱的分子间相互作用, 如静电和范德华力. 对于紧密捆绑德蛋白质而言, 后者尤其重要.

当前预测蛋白质构成的一种方法是求氨基酸的完整构造的最小势能值. 通过 Lennard-Jones 势能建立范德华力

$$U(r)=\frac{1}{r^{12}}-\frac{2}{r^6},$$

其中 r 代表两个原子之间的距离. 图 13-7 表明用势能定义能量源. 当 $r>1$ 时, 是吸引力; 当原子间距离比 $r=1$ 近时, 是很强的排斥力. 对一族原子, 它们的位置为 $(x_1,y_1,z_1),\cdots,(x_n,y_n,z_n)$, 被最小化的目标函数是关于所有原子对的两两 Lennard-Jones 势能

$$U=\sum_{i<j}\left(\frac{1}{r_{ij}^{12}}-\frac{2}{r_{ij}^6}\right)$$

之和, 其中

$$r_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2+(z_i-z_j)^2}$$

表示原子 i 和 j 的距离. 最优化问题中的向量是原子的直角坐标系. 要考虑平移对称和转动对称: 如果这一族沿着直线移动或转动, 总能量是不变的. 为了处理对称性, 我们将通过把第一个原子固定在原点 $v_1 = (0,0,0)$ 且要求第二个原子位于 z 轴上 $v_2 = (0,0,z_2)$, 来限制可能的构造. 剩余变量 $(x_3,y_3,z_3),\cdots,(x_n,y_n,z_n)$ 的位置是通过最小化势能函数 U 得到的.

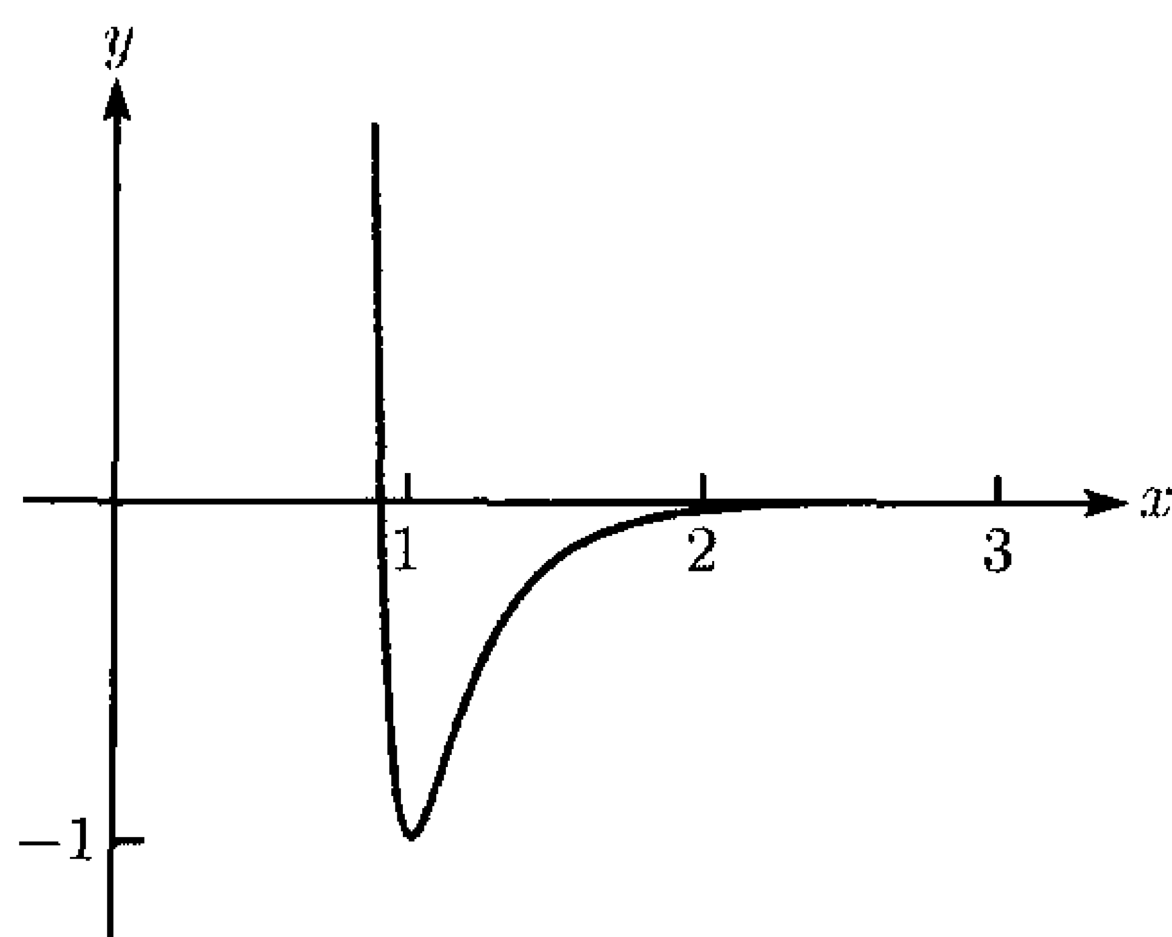


图 13-7 Lennard-Jones 势能 $U(r)=\frac{1}{r^{12}}-\frac{2}{r^6}$. 最小能量是 -1, 在 $r=1$ 处取得

在图 13-7 的帮助下, 通过计算 Lennard-Jones 能量的最小值来定位 4 个或更少的原子是简单的. 注意到单一势能在 $r=1$ 的最小势能是 -1. 因此, 两个原子可以相隔一个单位, 以至于能量位于槽的底部. 3 个原子可以置于等边三角形上, 第 4 个原子置于到这 3 个顶点距离相等处, 也就是, 位于这个三角形上方, 形成一个等边四面体. 对 $n=2,3,4$ 时的总势能 U 是 -1 与交互作用数的乘积, 或分别为 $-1,-3,-6$.

然而第 5 个原子的放置不是那么明显. 当 $n=4$ 时, 没有点到四面体的 4 个顶点是等距的, 所以需要新的方法: 数值优化.

活动建议

1. 写一个返回势能的函数文件. 用 Nelder-Mead 法去求解 $n=5$ 时的最小能量. 用不同的初始猜测, 直到你确信已经得到了绝对极小值. 需要多少步?
2. 用 MATLAB命令 `plot3`, 用圆圈画出 5 个原子的最小能量的构造, 且用线段连接所有的圆圈来观察符合要求的分子.
3. 延伸第 1 步中的函数使得它返回 f 和梯度向量 ∇f. 对 $n=5$ 的情况, 用最速下降法. 如前面那样求解最小能量.
4. 如果 MATLAB优化工具箱可行的话, 用命令 `fminunc`, 仅仅用目标函数 f.
5. 用 `fminunc`, 使用 f 和梯度 ∇f.
6. 把前面的方法作用于 $n=6$. 按照可靠性和有效性对方法分类.
7. 对更大的 n, 确定并画出它的最小能量构造. 当 n 大至几百时, 关于最小能量 Lennard-Jones 族的信息在许多网上都有提供, 因此你可以检查所得到的答案.

   蛋白质折叠问题已经成为各个学科优化研究的温床. 模拟退火和强有力的拟牛顿法经常用于预测复杂分子的构造, 同时针对分子间力进行更现实的建模. 蛋白质数据库 (Protein Data Bank) `http://www.rcsb.org/pdb` 对于生物大分子结构数据提供了有用的世界范围的存档. 通过实验, 更广泛地列出所测量的原子位置是可行的, 可用于测试和

确认关于力和能量最小化的假设.

软件和进一步阅读

关于最优化的介绍包括 [1,7,5]. 有用的指南 [4] 包含专门为优化而设计的许多软件包的参考. 大量的各种类型的测试问题可以在 [2] 中找到. 由西北大学和 Argonne 国家实验室运行的最优化技术中心 `http://www.ece.northwestern.edu/OTC` 有许多可用到软件的链接.

Netlib 的 `opt` 词典含有大量可行的优化方法, 包括: `hooke` (不带导数的无约束优化, 通过 Hooke 和 Jeeves 方法), `praxis` (不需要导数的无约束优化), 以及`tn` (无约束优化和简单约束优化的牛顿法). 由 Chapman 和 Naylor 提出的 WNLIB, 包含无约束和约束优化的非线性方法, 它是基于共轭梯度法和共轭方向法的 (以及一般的模拟退火算法).

MATLAB优化工具箱包括多种约束和无约束非线性优化问题的方法. TOMLAB 优化环境在 MATLAB工具箱的基础上提供了大量的非线性优化的工具. 它有统一的输入 — 输出格式、优选的 GUI 和对导数的自动处理. 在 `mathtool.net` 上的优化目录包括许多用 MATLAB和其他语言编写的解法.

附录A 矩阵代数

首先简要回顾一下矩阵代数中的基本定义.

A.1 矩阵基础

向量 (Vector)是一个数组

$$\boldsymbol{u}=\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}.$$

如果它含有 n 个数就称为 n 维向量. 我们常把前面竖直排列的数组即**列向量** (column vector) 与水平排列的数组即**行向量** (row vector)

$$\boldsymbol{u}=[u_1,u_2,\cdots,u_n]$$

区分开来. $m\times n$**矩阵** (matrix) 是一个 $m\times n$ 数组, 其形式为

$$\boldsymbol{A}=\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

$\boldsymbol{A}$ 的每一行 (水平的) 可以认为是 $\boldsymbol{A}$ 的一个行向量, 而每一列 (竖直的) 可以认为是 $\boldsymbol{A}$ 的一个列向量.

矩阵 – 向量乘法产生一个新向量, 定义如下:

$$\boldsymbol{Au}=\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}=\begin{bmatrix} a_{11}u_1+a_{12}u_2+\cdots+a_{1n}u_n \\ \vdots \\ a_{m1}u_1+a_{m2}u_2+\cdots+a_{mn}u_n \end{bmatrix}. \tag{A.1}$$

注意, 为了用一个 d 维向量去乘 $m\times n$ 矩阵, 必需 $n=d$.

在矩阵 – 矩阵乘法中, 一个 $m\times n$ 矩阵乘以一个 $n\times p$ 矩阵得到的乘积是 $m\times p$ 矩阵. 矩阵相乘能用矩阵 – 向量乘法来表示. 令 $\boldsymbol{C}$ 是 $n\times p$ 矩阵, 把它写成列向量形式

$$\boldsymbol{C}=[\boldsymbol{c}_1|\cdots|\boldsymbol{c}_p].$$

那么矩阵 $\boldsymbol{A}$ 和矩阵 $\boldsymbol{C}$ 的乘积就是

$$\boldsymbol{AC}=\boldsymbol{A}[\boldsymbol{c}_1|\cdots|\boldsymbol{c}_p]=[\boldsymbol{Ac}_1|\cdots|\boldsymbol{Ac}_p].$$

可以把含 n 个未知量, m 个线性方程的方程组写成矩阵形式

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

称为**矩阵方程** (matrix equation).

$n \times n$ **单位矩阵** (identity matrix) $\boldsymbol{I}_n$ 是 $I_{ii} = 1, I_{ij} = 0$ $(1 \leqslant i, j \leqslant n; i \neq j)$ 的矩阵. 对于矩阵的乘法运算, 单位矩阵的作用就像单位一样, 即 $\boldsymbol{AI}_n = \boldsymbol{I}_n\boldsymbol{A} = \boldsymbol{A}$, 对任一 $n \times n$ 矩阵 $\boldsymbol{A}$ 都成立. 对于 $n \times n$ 矩阵 $\boldsymbol{A}$, $\boldsymbol{A}$ 的**逆** (inverse) $\boldsymbol{A}^{-1}$ 是一个 $n \times n$ 矩阵, 它满足 $\boldsymbol{AA}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n$. 如果 $\boldsymbol{A}$ 有逆矩阵, 就称它为**可逆的** (invertible). 不可逆的矩阵称为**奇异的** (singular).

$m \times n$ 矩阵 A 的**转置** (transpose)是矩阵 $\boldsymbol{A}^{\mathrm{T}}$, 其元素为 $A_{ij}^{\mathrm{T}} = A_{ji}$. 乘积的转置法则是 $(\boldsymbol{AB})^{\mathrm{T}} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}$.

有两种方式进行两个向量的相乘. 设

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}, \quad \boldsymbol{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}.$$

内积 (inner product) $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v}$, 先把 $\boldsymbol{u}$ 转置成行向量, 然后由通常的矩阵乘法给出

$$\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} = u_1v_1 + \cdots + u_nv_n.$$

因此 $1 \times n$ 和 $n \times 1$ 矩阵的乘积得到的结果是 1×1 矩阵 (或者实数). 如果 $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} = 0$, 那么两个列向量**垂直** (orthogonal). **外积** (outer product) $\boldsymbol{uv}^{\mathrm{T}}$ 是 $n \times 1$ 列乘以 $1 \times n$ 行. 通常的矩阵乘法就给出一个 $n \times n$ 矩阵, 其结果是

$$\boldsymbol{uv}^{\mathrm{T}} = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & & \vdots \\ u_nv_1 & \cdots & \cdots & u_nv_n \end{bmatrix}.$$

外积是秩为 1 的矩阵.

因为计算逆矩阵的计算复杂性高, 所以要尽可能避免或减少这种计算. Sherman-Morrison 公式是一种有益的技巧. 假设已经知道 $n \times n$ 矩阵 $\boldsymbol{A}$ 的逆矩阵, 并且我们需要修改后的矩阵 $\boldsymbol{A} + \boldsymbol{uv}^{\mathrm{T}}$ 的逆矩阵, 这里 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 是 n 维向量.

定理 A.1(Sherman-Morrison 公式) *如果 $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{u} \neq -1$, 那么 $\boldsymbol{A} + \boldsymbol{uv}^{\mathrm{T}}$ 可逆并且*

$$(\boldsymbol{A} + \boldsymbol{uv}^{\mathrm{T}})^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{uv}^{\mathrm{T}}\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{u}}.$$

通过把 $\boldsymbol{A} + \boldsymbol{uv}^{\mathrm{T}}$ 乘以公式中的表达式就得到 Sherman-Morrison 公式. 因为 $\boldsymbol{uv}^{\mathrm{T}}$ 是秩为 1 的矩阵, 所以 $\boldsymbol{A} + \boldsymbol{uv}^{\mathrm{T}}$ 称为 $\boldsymbol{A}$ 的秩 1 校正 (关于 Sherman-Morrson 公式的一种重要应用, 可见第 2 章中 Broyden 方法的讨论. 能在线性代数教科书, 比如 [4, 2] 中找到有关矩阵的基本内容).

A.2 分块相乘

矩阵乘法可以分块进行, 这在第 12 章中将非常有益. 如果把两个矩阵分成与矩阵乘法相容的小块, 那么就能用块状矩阵乘法来进行这种矩阵的求积. 例如, 可以用以下分块来进行两个 3×3 矩阵的乘积:

$$\boldsymbol{AB} = \left[\begin{array}{c:cc} x & x & x \\ \hdashline x & x & x \\ x & x & x \end{array}\right] \left[\begin{array}{c:cc} x & x & x \\ \hdashline x & x & x \\ x & x & x \end{array}\right] = \left[\begin{array}{c:c} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \hdashline \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{array}\right] \left[\begin{array}{c:c} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \hdashline \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{array}\right]$$

$$= \left[\begin{array}{c:c} \boldsymbol{A}_{11}\boldsymbol{B}_{11} + \boldsymbol{A}_{12}\boldsymbol{B}_{21} & \boldsymbol{A}_{11}\boldsymbol{B}_{12} + \boldsymbol{A}_{12}\boldsymbol{B}_{22} \\ \hdashline \boldsymbol{A}_{21}\boldsymbol{B}_{11} + \boldsymbol{A}_{22}\boldsymbol{B}_{21} & \boldsymbol{A}_{21}\boldsymbol{B}_{12} + \boldsymbol{A}_{22}\boldsymbol{B}_{22} \end{array}\right]$$

这里 $\boldsymbol{A}_{11}$ 和 $\boldsymbol{B}_{11}$ 是 1×1 矩阵, $\boldsymbol{A}_{12}$ 和 $\boldsymbol{B}_{12}$ 是 1×2 矩阵, 如此等等. 例如

$$\left[\begin{array}{c|cc} 1 & 2 & 3 \\ \hline 0 & 1 & 3 \\ 2 & 2 & 4 \end{array}\right]\left[\begin{array}{c|cc} 2 & 4 & 1 \\ \hline 1 & 0 & 1 \\ 3 & 1 & 2 \end{array}\right] = \left[\begin{array}{cc} 1\times 2+[2\ 3]\begin{bmatrix}1\\3\end{bmatrix} & 1[4\ 1]+[2\ 3]\begin{bmatrix}0&1\\1&2\end{bmatrix} \\ \begin{bmatrix}0\\2\end{bmatrix}2+\begin{bmatrix}1&3\\2&4\end{bmatrix}\begin{bmatrix}1\\3\end{bmatrix} & \begin{bmatrix}0\\2\end{bmatrix}[4\ 1]+\begin{bmatrix}1&3\\2&4\end{bmatrix}\begin{bmatrix}0&1\\1&2\end{bmatrix} \end{array}\right]$$

$$= \left[\begin{array}{c|cc} 13 & 7 & 9 \\ \hline 10 & 3 & 7 \\ 18 & 12 & 12 \end{array}\right].$$

进行分块矩阵乘法给出与没有分块的乘法相同的结果. 这个着眼于矩阵乘法的另一种方式不但减少了计算, 而且帮助簿记, 特别是在第 12 章的特征值的计算.

分块所需要的相容性是 $\boldsymbol{A}$ 的列数必须刚好与 $\boldsymbol{B}$ 的行数相匹配. 在前面的例子中, $\boldsymbol{A}$ 的第一列是一组, 而最后两列是另一组. 对矩阵 $\boldsymbol{B}$, 第一行是一组, 而最后两行是另一组. 再举一个例子, 我们能分块进行 3×5 矩阵 $\boldsymbol{A}$ 和 5×2 矩阵 $\boldsymbol{B}$ 的乘法:

$$\left[\begin{array}{c|ccc|c} x & x & x & x & x \\ x & x & x & x & x \\ \hline x & x & x & x & x \end{array}\right]\left[\begin{array}{c|c} x & x \\ \hline x & x \\ x & x \\ x & x \\ \hline x & x \end{array}\right] = \left[\begin{array}{c|c|c} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{A}_{13} \\ \hline \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \boldsymbol{A}_{23} \end{array}\right]\left[\begin{array}{c|c} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \hline \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \\ \hline \boldsymbol{B}_{31} & \boldsymbol{B}_{33} \end{array}\right]$$

$$= \left[\begin{array}{c|c} A_{11}B_{11}+A_{12}B_{21}+A_{13}B_{31} & A_{11}B_{12}+A_{12}B_{22}+A_{13}B_{32} \\ \hline A_{21}B_{11}+A_{22}B_{21}+A_{23}B_{31} & A_{21}A_{12}+A_{22}B_{22}+A_{23}B_{32} \end{array}\right].$$

在这一情形, $\boldsymbol{A}$ 的列分成三组, $\boldsymbol{B}$ 的行分成三组, 正好相匹配. 另一方面, $\boldsymbol{A}$ 的行的分组与 $\boldsymbol{B}$ 的列的分组不需匹配, 它们可以任意操作.

A.3 特征值和特征向量

我们首先简单回顾一下特征值和特征向量的基本概念.

定义 A.2 *设 $\boldsymbol{A}$ 是 $m\times m$ 矩阵, $\boldsymbol{x}$ 是 m 维实或复的非零向量. 如果对某一实数或复数 λ 成立 $\boldsymbol{Ax}=\lambda\boldsymbol{x}$, 那么称 λ 为 A 的***特征值** (eigenvalue), *而 x 是相应的***特征向量** (eigenvector).

例如, 矩阵 $\boldsymbol{A}=\begin{bmatrix}1&3\\2&2\end{bmatrix}$ 有特征向量 $\begin{bmatrix}1\\1\end{bmatrix}$, 相应的特征值是 4.

特征值是**特征值多项式** (characteristic polynomial)$\det(\boldsymbol{A}-\lambda\boldsymbol{I})$ 的根 λ. 若 λ 是 $\boldsymbol{A}$ 的特征值, 则 $\boldsymbol{A}-\lambda\boldsymbol{I}$ 的零空间中的任意非零向量就是对应于 λ 的特征向量. 对这个例子,

$$\det(\boldsymbol{A}-\lambda\boldsymbol{I})=\det\begin{bmatrix}1-\lambda&3\\2&2-\lambda\end{bmatrix}=(\lambda-1)(\lambda-2)-6=(\lambda-4)(\lambda+1), \tag{A.2}$$

所以特征值是 $\lambda=4,-1$. 相应于 $\lambda=4$ 的特征向量在

$$\boldsymbol{A}-4\boldsymbol{I}=\begin{bmatrix}-3&3\\2&-2\end{bmatrix} \tag{A.3}$$

的零空间求得, 因此由 $\begin{bmatrix}1\\1\end{bmatrix}$ 的非零倍数组成. 类似地, 相应于 $\lambda=-1$ 的特征向量是 $\begin{bmatrix}3\\-2\end{bmatrix}$ 的所有的非零倍数.

定义 A.3 *如果存在可逆 $m\times m$ 矩阵 $\boldsymbol{S}$, 使得 $\boldsymbol{A}_1=\boldsymbol{S}\boldsymbol{A}_2\boldsymbol{S}^{-1}$, 就称 $m\times m$ 矩阵 $\boldsymbol{A}_1$ 与 $\boldsymbol{A}_2$ 相似* (similar), *记作 $\boldsymbol{A}_1\sim\boldsymbol{A}_2$.*

因为相似矩阵的特征多项式相同，所以它们有相同的特征值：

$$\boldsymbol{A}_1 - \lambda \boldsymbol{I} = \boldsymbol{S}\boldsymbol{A}_2\boldsymbol{S}^{-1} - \lambda \boldsymbol{I} = \boldsymbol{S}(\boldsymbol{A}_2 - \lambda \boldsymbol{I})\boldsymbol{S}^{-1} \tag{A.4}$$

意味着

$$\det(\boldsymbol{A}_1 - \lambda \boldsymbol{I}) = (\det \boldsymbol{S})\det(\boldsymbol{A}_2 - \lambda \boldsymbol{I})\det \boldsymbol{S}^{-1} = \det(\boldsymbol{A}_2 - \lambda \boldsymbol{I}). \tag{A.5}$$

如果矩阵 $\boldsymbol{A}$ 有形成 $\mathbf{R}^m$ 的基的特征向量，那么，$\boldsymbol{A}$ 与对角矩阵相似，并且称 $\boldsymbol{A}$ 是**可对角化的** (diagonalizable). 事实上，假设 $\boldsymbol{A}\boldsymbol{x}_i = \lambda_i \boldsymbol{x}_i (i = 1, \cdots, m)$ 并且定义矩阵

$$\boldsymbol{S} = [\boldsymbol{x}_1 \ \cdots \ \boldsymbol{x}_m],$$

那么可以验证矩阵方程

$$\boldsymbol{A}\boldsymbol{S} = \boldsymbol{S}\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_m \end{bmatrix} \tag{A.6}$$

成立. 因为矩阵 $\boldsymbol{S}$ 的列 (向量) 张成 $\mathbf{R}^m$，所以 $\boldsymbol{S}$ 可逆. 因此 $\boldsymbol{A}$ 与一个包含其特征值的对角矩阵相似.

即使在 2×2 情形，也不是所有的矩阵可以对角化. 事实上，所有 2×2 矩阵相似于以下三种类型之一：

$$\boldsymbol{A}_1 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}, \quad \boldsymbol{A}_2 = \begin{bmatrix} a & 1 \\ 0 & a \end{bmatrix}, \quad \boldsymbol{A}_3 = \begin{bmatrix} a & -b \\ b & a \end{bmatrix}.$$

请记住：相似矩阵的特征值相同. 如果有张成 $\mathbf{R}^2$ 的两个特征向量，那么 $\boldsymbol{A}$ 与第一种情形相似；如果有重特征值并仅有一维特征向量空间，那么矩阵与第二种情形相似；如果有一对复特征值，那么就与第三种情形相似.

A.4 对称矩阵

对于对称矩阵，所有特征向量互相正交，并且它们一起张成基础空间. 换言之，对称矩阵有一组正交特征向量基.

定义 A.4 *如果向量组的成员是彼此正交的单位向量，则这组向量是正交的.*

用点积表示向量组 $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m\}$ 的正交性就是：如果 $i \neq j$，则 $\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_j = 0$，并且 $\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_i = 1(1 \leqslant i, j \leqslant m)$. 例如，向量组 $\{(1,0,0),(0,1,0),(0,0,1)\}$ 和 $\{(\sqrt{2}/2, \sqrt{2}/2), (\sqrt{2}/2, -\sqrt{2}/2)\}$ 是正交向量组.

定理 A.5 *假设 $\boldsymbol{A}$ 是实对称 $m \times m$ 矩阵. 那么特征值是实数，并且 $\boldsymbol{A}$ 的单位特征向量组是形成 $\mathbf{R}^m$ 基的正交向量组 $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m\}$.*

例 A.1 求

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix}. \tag{A.7}$$

的特征值和特征向量.

如前计算，特征值/特征向量对是 $2, (1,2)^{\mathrm{T}}$ 和 $-1/2, (-2,1)^{\mathrm{T}}$. 注意，如定理所指出的，特征向量正交. 相应的正交单位特征向量是

$$\left\{ \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix}, \begin{bmatrix} -\frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix} \right\}.$$

◀

以下定理将有助于学习第 2 章中的迭代方法：

定义 A.6 *方阵 $\boldsymbol{A}$ 的**谱半径** $\rho(\boldsymbol{A})$ 是它的特征值的模的最大值.*

定理 A.7 如果 $n \times n$ 矩阵 $\boldsymbol{A}$ 的谱半径 $\rho(\boldsymbol{A}) < 1$, $\boldsymbol{b}$ 是任意向量, 那么对任一向量 $\boldsymbol{x}_0$, 迭代 $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b}$ 收敛. 实际上, 存在唯一向量 $\boldsymbol{x}_*$ 使得 $\lim_{k \to \infty} \boldsymbol{x}_k = \boldsymbol{x}_*$ 而且 $\boldsymbol{x}_* = \boldsymbol{A}\boldsymbol{x}_* + \boldsymbol{b}$.

而且, 如果 $\boldsymbol{b} = \boldsymbol{0}$, 那么 $\boldsymbol{x}_*$ 或者是零向量或者是 $\boldsymbol{A}$ 的关于特征值 1 的特征向量. 因为谱半径 $\rho(\boldsymbol{A}) < 1$. 后者不予考虑, 这就导致以下在第 8 章中有用的事实.

推论 A.8 如果 $n \times n$ 矩阵 $\boldsymbol{A}$ 的谱半径 $\rho(\boldsymbol{A}) < 1$, 那么, 对任一初始向量 $\boldsymbol{x}_0$, 迭代 $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k$ 收敛到 $\boldsymbol{0}$.

A.5 向量微积分

本节定义标量值和向量值函数的导数, 而且为了以后的使用还收集了它们的乘积法则.

设 $f(x_1, \cdots, x_n)$ 是 n 个变量的标量值函数. f 的**梯度** (gradient) 是向量值函数

$$\nabla f(x_1, \cdots, x_n) = [f_{x_1}, \cdots, f_{x_n}],$$

这里下标表示 f 关于那个变量的偏导数.

设

$$\boldsymbol{F}(x_1, \cdots, x_n) = \begin{bmatrix} f_1(x_1, \cdots, x_n) \\ \vdots \\ f_n(x_1, \cdots, x_n) \end{bmatrix}$$

是 n 个变量的向量值函数. $\boldsymbol{F}$ 的 Jacobi 矩阵是

$$\mathrm{D}\boldsymbol{F}(x_1, \cdots, x_n) = \begin{bmatrix} \nabla f_1 \\ \vdots \\ \nabla f_n \end{bmatrix}.$$

现在可以叙述在矩阵代数中两种典型积的乘积法则. 当把它们写成分量并且应用单变量乘积法则时, 两者都有简洁的证明. 设 $\boldsymbol{u}(x_1, \cdots, x_n)$ 和 $\boldsymbol{v}(x_1, \cdots, x_n)$ 是向量值函数, 而且令 $\boldsymbol{A}(x_1, \cdots, x_n)$ 是 $n \times n$ 矩阵函数. 点积 $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v}$ 是标量值函数. 下面第一个公式表示如何求它的梯度. 矩阵和向量的乘积 $\boldsymbol{A}\boldsymbol{v}$ 是一个向量, 它的 Jacobi 矩阵表示由第二条法则给出.

向量点积法则

$$\nabla(\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v}) = \boldsymbol{v}^{\mathrm{T}}\mathrm{D}\boldsymbol{u} + \boldsymbol{u}^{\mathrm{T}}\mathrm{D}\boldsymbol{v}$$

矩阵/向量乘积法则

$$\mathrm{D}(\boldsymbol{A}\boldsymbol{v}) = \boldsymbol{A} \cdot \mathrm{D}\boldsymbol{v} + \sum_{i=1}^{n} \boldsymbol{v}_i \mathrm{D}\boldsymbol{a}_i,$$

这里 $\boldsymbol{a}_i$ 表示 $\boldsymbol{A}$ 的第 i 列.

附录B　MATLAB简介

MATLAB是一个普遍适用的计算软件尤其适用于执行数学方法和数值方法. 对于小型问题来说, MATLAB是一个高能的计算器; 对于大型复杂问题来说, 则是一种功能完备的编程语言. MATLAB 的一个有益之处在于它大量的高质量的库函数, 这些函数可以使复杂的计算变得简洁明了, 并且容易用高阶的代码写出来.

本节包括对 MATLAB命令及特性的简要介绍. 更详细的内容可以在 MATLAB的帮助文件、MATLAB用户指南、参考书 [3, 1] 以及致力于程序包的网站里找到.

B.1　启动 MATLAB

在基于 PC 的系统中, 启动 MATLAB只需双击相应的图标, 退出只要依次点击菜单中的 "文件/退出". 而在基于 UNIX 的系统中, 在系统提示符处输入 MATLAB代码 `$matlab`, 然后输入 `>> exit` 退出 MATLAB. 输入命令 `>> a=5`, 接着按回车键, MATLAB会把信息反馈给你. 输入命令

```
>> b=3
>> c=a+b
>> c=a*b
>> d=log(c)
>> who
```

会帮助你认识 MATLAB是如何工作的. 如果想停止反馈信息, 可以在每条命令后面加上一个分号. `who` 命令会出现一个列表, 显示出所定义的所有变量.

MATLAB有着丰富的在线帮助功能. 输入 `help log` 将会获取有关 `log` 命令的信息. MATLAB的 PC 版本有一个 "帮助" 菜单, 其中含有关于所有命令的描述和使用建议.

想消去变量 a 的值, 输入 `clear a`. 输入 `clear` 会消去所有先前定义的变量. 想找回此前输入过的命令, 按 "上箭头" 键. 如果命令太长, 在一行里输不下, 在行末尾输入 3 个点号并回车, 然后在下一行接着输入命令.

为了下次登录 MATLAB还能使用这些数据, 我们需要保存变量的值, 输入 `save`; 下次运行 MATLAB的时候, 只要输入 `load` 就可以恢复它们. 想记录 MATLAB会话窗口的部分或完全内容, 输入 `diary filename` 开始记录, 输入 `diary off` 结束记录. 用你选择的文件名来代替 `filename`. 这个命令有助于把你当前的工作作为作业提交. `diary` 命令会产生一个文件, 这个文件在 MATLAB会话结束之后可以查看或者打印.

MATLAB一般在 IEEE 双精度下 (大约有 16 位十进制数的精度) 执行所有的计算. 数值显示格式可以用 `format` 语句予以改变. 输入 `format long` 会改变数字显示的方式, 直到下个 `format` 命令输入. 例如, 1/3 这个数, 格式不一样, 显示也不一样:

```
 format short      0.3333 format short e
3.3333E-001 format long       0.33333333333333 format long e
3.333333333333333E-001 format bank      0.33 format hex
3fd5555555555555
```

想要控制更多的输出格式, 就要用 `fprintf` 命令. 命令

```
>> x=0:0.1:1;
>> y=x.^2;
>> fprintf('%8.5f %8.5f \n',[x;y])
```

将会打印出下面这个表格



```
0.00000  0.00000
0.10000  0.01000
0.20000  0.04000
0.30000  0.09000
0.40000  0.16000
0.50000  0.25000
0.60000  0.36000
0.70000  0.49000
0.80000  0.64000
0.90000  0.81000
1.00000  1.00000
```

B.2 绘　　图

要想对数据进行绘图, 先要把它们表示成 X、Y 方向的向量形式. 例如, 命令

```
>> a=[0.0 0.4 0.8 1.2 1.6 2.0];
>> b=sin(a);
>> plot(a,b)
```

将会画出 $y = \sin x, 0 \leqslant x \leqslant 2$ 的一条逐段线性近似的图像, 如图 B-1 a 所示. 本例中, a和 b 是 6 维向量, 或者说 6 元数组. 数轴上数的字号可以设为 16 点, 例如, 可以输入这个命令 set(gca, 'FontSize',16). 一个更简单的方法是用下面的命令定义向量 a:

```
>> a=0:0.4:2;
```

这个命令定义向量 a 的首个分量是 0, 递增的步长为 0.4, 最后一个分量是 2, 与前面的那种较长的定义相同. 要绘出正弦曲线更精细的图像, 可以用下面的命令:

```
>> a=0:0.02:2*pi;
>> b=sin(a);
>> plot(a,b)
```

结果如图 B-1 b 所示.

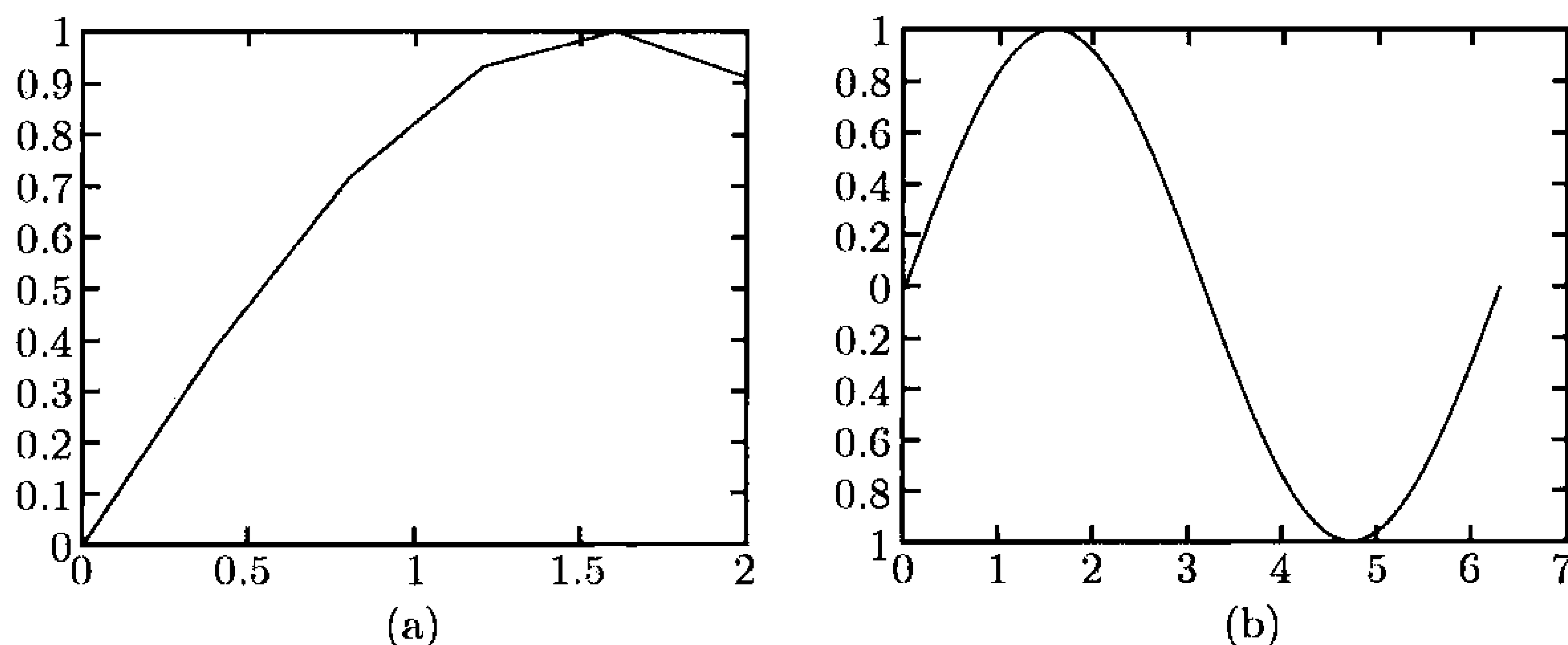


图 B-1　MATLAB图像. (a) $f(x) = \sin x$ 的逐段线性图像, 其中 x 的步长增量为 0.4. (b) 另一个逐段图像, 看起来更光滑一些, 那是因为 x 的步长增量是 0.02

想要绘出 $y = x^2, 0 \leqslant x \leqslant 2$ 的图像, 可以用下面的命令:

```
>> a=0:0.02:2;
>> b=a.^2;
>> plot(a,b)
```

上面的命令中有个奇怪而意想不到的符号. 幂次符号前面的点号是让 MATLAB对向量 a 的每一个元素进行

平方运算. 就如我们将在 B.3 节中看到的那样, MATLAB把每一个变量都视为矩阵或者双指标数组. 如果略去幂次符号前面的点号, 意味着将把 101 × 1 的矩阵 **a** 自乘, 按照矩阵相乘的法则, 这是不可乘的. 如果你非要这么做, MATLAB就会报错. 一般来说, MATLAB把 "带点号的运算" 看成是对每个元素逐个进行的, 而不是矩阵相乘.

下面介绍更高阶的绘图技巧. 如果不特别指明, MATLAB会自动选择数轴上的标度, 就像在图 B-1 中那样. 要想手动选择数轴上的标度, 就要用 **axis** 命令. 例如, 下面的命令

```
>> v=[-1 1 0 10 ]; axis(v)
```

把图像窗口设为 $[-1,1]\times[0,10]$. **grid** 命令会在图像后面绘制网格.

用命令 **plot(x1, y1, x2, y2, x3, y3)** 可以在同一个图像窗口绘出 3 条曲线, 每对 **xi**、**yi** 绘制一条曲线. 输入 **help plot** 将会看到实线图、点线图、虚线图类型, 以及不同的数据点形状 (圆形、点形、三角形、正方形等) 的选择. 用 **semilogy** 和 **semilogx** 命令可以绘制 "半对数" 图像.

subplot 命令可以把图像窗口分割成多个部分. **subplot(abc)** 命令把窗口分成 $a\times b$ 个, 当前图像使用第 c 个窗口. 例如,

```
>> subplot(121),plot(x,y)
>> subplot(122),plot(x,z)
```

把第一个图像绘制在屏幕左边, 第二个绘制在右边. **figure** 命令会打开新的绘图窗口并在不同的绘图窗口间切换. 当你需要同时查看几个不同的图像时, 就需要用到这个命令.

三维图的绘制要使用 **mesh** 命令. 例如, 在定义域 $[-1,1]\times[-2,2]$ 上绘制 $z=\sin(x^2+y^2)$ 的图像, 就要用下面的命令:

```
>> [x,y]=meshgrid(-1:0.1:1,-2:0.1:2);
>> z=sin(x.^2+y.^2);
>> mesh(x,y,z)
```

由 **meshgrid** 命令产生的向量 **x** 是具有 21 行的 21 维行向量 **-1:0.1:1**, 类似地, 向量 **y** 是具有 41 列的列向量 **-2:0.1:2**. 由这些代码产生的图像如图 B-2 所示. 用 **surf** 代替 **mesh** 命令, 则会在图像的表面添上颜色.

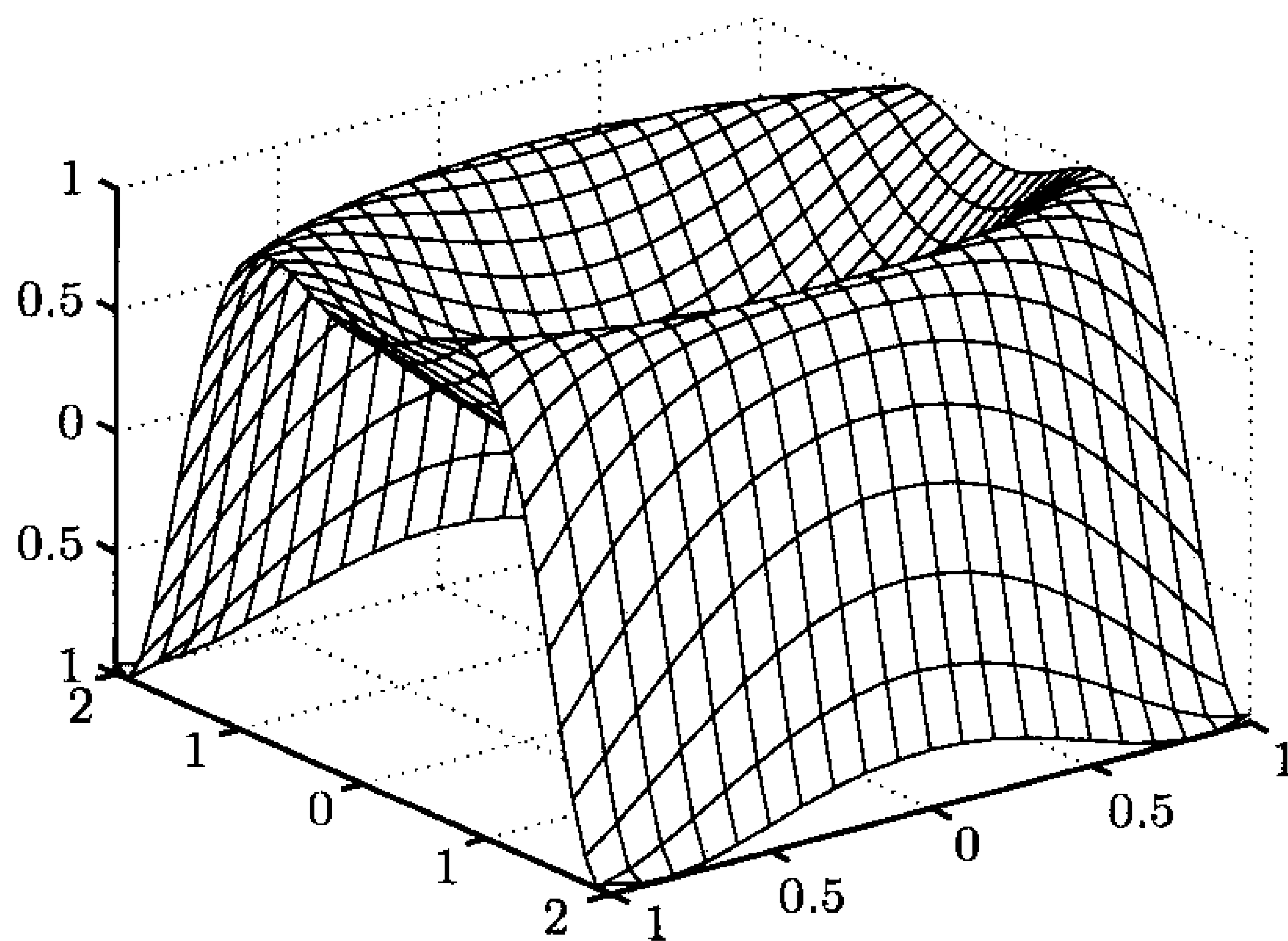


图 B-2　三维 MATLAB图像. **mesh** 命令用于绘制三维图的表面

B.3　MATLAB程序

用 MATLAB语言编写程序能获得更精细的结果. .m 文件包含了 MATLAB的一系列命令. .m 文件的名称有个后缀 "**.m**". 例如, 你可以用个人偏好的编辑器或者 MATLAB自带的编辑器去创建文件**cubrt.m**, 其中包含了下面的命令:

```
% The program cubrt.m finds a cube root by iteration
y=1;
n=15;
z=input('Enter z:');
for i = 1:n
  y = 2*y/3 + z/(3*y^2)
end
```

要运行程序, 只要在 MATLAB提示符处输入 **cubrt**. 这些代码的结果将收敛到 z 的三次方根, 其中的原理在你学过第 1 章的牛顿方法以后就已经很清楚了. 注意到, 分号在由迭代定义新的 **y** 那一行的末尾被省略了. 这会让你看到 **y** 逐步逼近 **z** 的三次方根的近似过程.

有了 MATLAB的图像功能, 我们就可以分析三次方根算法中的数据. 考虑程序 **cubrt1.m** :

```
% The program cubrt1.m finds cube roots and displays its progress
y(1)=1;
n=15;
z=input('Enter z:');
for i = 1:n-1
  y(i+1) = 2*y(i)/3 + z/(3*y(i)^2);
end
plot(1:n,y)
title('Iterative method for cube roots')
xlabel('Iteration number')
ylabel('Approximate cube root')
```

在上面的程序里, 以 $z = 64$ 运行一下. 完成时, 输入下面的命令:

```
>> e=y-4;
>> plot(1:n,e)
>> semilogy(1:n,e)
```

第一条命令从向量 **y** 的每一个元素中减去正确的三次方根 4, 这个差就是每步迭代产生的误差**e**. 第二条命令描绘出了误差的图像. 第三条命令描绘出了误差的"半对数图像"——即在 y 方向使用了对数单位.

B.4 流程控制

for 循环在前面求三次方根的程序里已经介绍过了. MATLAB有若干命令用于控制程序的流程. 大部分的命令, 包括 **while** 循环、**if** 和 **break** 语句, 对于那些了解高级编程语言的读者来说应该比较熟悉. 例如,

```
n=5;
 for i=1:n
  for j=1:n
    a(i,j)=1/(i+j-1);
  end
end
a
```

生成并显示了一个 5×5 的 Hilbert 矩阵. 分号避免了重复显示一些中间过程的结果. 最后把最终结果 **a** 显示出来. 注意到每个 **for** 命令必须和一个 **end** 命令匹配. 虽然使用缩进格式在 MATLAB里不是必需的, 但是这有利于大大提高程序的可读性.

while 命令运行的原理类似:

```
n=5;i=1;
while i<=n
  j=1;
  while j<=n
    a(i,j)=1/(i+j-1);
    j=j+1;
  end
  i=i+1;
end
a
```

上面的命令产生了和两个 for 循环相同的结果.

if 命令用于确定流程, 此时 break 命令能退出下一次的内循环. 下面的程序将阐明这两个命令的使用方法:

```
% To compute the nth derivative of sin(x) at x=0
n=input('Enter n, negative number to quit:)
if n<=0,break,end
r=rem(n,4)  % rem is the remainder function
if r==0
  y=0
elseif r==1
    y=1
elseif r==2
    y=0
else
    y=-1
end
y
```

逻辑运算符 & 和 | 分别代表逻辑与 (AND) 和逻辑或 (OR) 关系. error 命令停止执行.m 文件, 并把相关信息告知用户.

B.5 函　　数

如果计算所用的程序不仅仅是几行的话, 那么创造一个.m 文件去保存 MATLAB代码是个更好的选择. 一个.m 文件可以调用若干.m 文件, 也可以调用自身. (输入 ⟨Ctrl⟩+C 将通常用于跳出 MATLAB的死循环)

在 MATLAB中, 函数是一类很特别的.m 文件, 它可以传递值. 就像下面的例子那样, 第一行的语法必须遵守, 文件名是f.m:

```
function y=f(x)
% Evaluates the function sin(log(x)), if it makes sense,
% otherwise returns zero.
if x>0
  y=sin(log(x));
else
  y=0
end
```

函数与普通.m 文件的区别之处仅仅在第一行. 文件名, 除去 ".m" 之外, 必须与第一行里的函数名称一致. 因此, MATLAB里的函数是一种特殊的.m 文件. 函数文件里的变量默认为局部变量, 但是可以由 global 命令声明为全局变量.

更复杂的程序可以有多个变量作为输入, 多个变量作为输出. 例如, 这个程序调用 MATLAB自带的函数 mean 和 std, 然后把它们的结果以数组的形式输出:

```
function [m,sigma]=stat(x)
% Returns sample mean and standard deviation of input vector x
m=mean(x);
sigma=std(x);
```

如果 stat.m 这个文件在 MATLAB 的当前路径上, 那么输入 stat(x), 其中 x 是一个向量, 将会返回输入向量 x 的均值和标准差.

nargin 命令可以返回函数输入参数的个数. 有了这个命令, 函数的工作会被改变, 取决于出现在该函数中参数数量. nagin 命令的一个例子在关于嵌套乘法的程序 0.1 中给出.

在 MATLAB里, 一个函数调用别的函数有很多方法. 为了尽可能说清楚, 这本书里我们通常使用 "硬调用" 的办法, 也就是在调用函数里包含了被调用函数的函数定义. 如果我们的目标是求 $\sin 2x$ 在 $x = 0$ 的

近似导数值, 就可以建立一个叫 deriv.m 的文件, 文件采用了第 5 章中的计算方法.

```
function y=deriv(x,h)
% Returns derivative approximation at x with step size h
y=(f(x+h)-f(x-h))/(2*h);

function y=f(x)
y=sin(2*x);
```

然后输入这行命令

```
>> deriv(0, 0.0001)
```

得到的是近似的结果. 函数 `f` 可以定义在 `deriv.m` 里面或者作为一个单独的 `f.m` 文件. 这种方法显而易见, 但是不够巧妙. 因为在改变输入函数的时候, `deriv.m` 或者 `f.m` 文件都需要改变.

更灵活的办法是输入一个所谓的**内联函数** (inline function). 例如, 如下创建一个 `deriv1.m` 文件:

```
function y=deriv1(f,x,h)
% Returns derivative approximation at x with step size h
y=(f(x+h)-f(x-h))/(2*h);
```

注意到函数现在依然作为输入参数. 那么下面的命令

```
>> f=inline('sin(2*x)','x');
>> deriv1(f,0,0.0001)
```

将得到近似的导数值. 由这种方法, 很容易改变 `f`, 此外, 还强调了这里函数 `f` 可以被视为输入参数.

为了防止函数 $f(x)$ 过于复杂而不能在同一行里写下, 一种简便的方法就是声明为内联函数. 我们可以运行任何一个函数文件, 例如 `f1.m`,

```
function y=f1(x)
x=2*x;
y=sin(x);
```

然后输入命令

```
>> f=inline('f1(x)','x');
>> deriv1(f,0,0.0001)
```

就得到了导数的近似值. 我们鼓励读者使用内联的思想, 因为这更有利于写出更高效、更灵活的代码.

B.6 矩阵运算

实现 MATLAB强大的多功能性的关键在于变量数据结构的复杂性. MATLAB中的每个变量都是一个其元素是双精度浮点数的 $m \times n$ 矩阵. 单个数则是特殊的 1×1 矩阵. 下面的格式

```
>> A=[1 2 3
4 5 6]
```

或者

```
>> A=[1 2 3; 4 5 6]
```

定义了一个 2×3 矩阵 $\boldsymbol{A}$. 命令 `B = A'`创建了一个 3×2 矩阵 B, 它是 A 的转置. 维数相同的矩阵可以用+和−运算符实现加减. 命令 `size(A)` 返回矩阵 A 的维度, 而 `length(A)` 返回两个维度当中比较大的那个.

MATLAB提供了许多能轻松创建矩阵的命令. 例如 `zeros(m, n)` 产生一个 $m \times n$ 全零矩阵. 如果 A 是一个矩阵, 那么 `zeros(size(A))` 产生一个和 A 维数一样的全零阵. 命令 `ones(m, n)` 和 `eye(m, n)` (产生的是单位矩阵) 也有类似的用法. 例如,

```
>> A=[eye(2) zeros(2, 2); zeros(2, 2) eye(2)]
```

是创建 4×4 单位矩阵的一个复杂的、但却是正确的办法.

冒号运算符能从一个矩阵里抽取出子矩阵. 例如,

```
>> b=A(1:3, 2)
```
把 A 矩阵的第二列的前三个元素赋予 b. 命令
```
>> b= A(:, 2)
```
把 A 矩阵的整个第二列赋予 b. 命令
```
>> B=A(:, 1:3)
```
把 A 矩阵的前三列子矩阵赋予 B 矩阵.

$m \times n$ 矩阵 A 和 $n \times p$ 矩阵 B 可以由命令 C=A*B 执行相乘. 如果矩阵的维数不合适, 那么 MATLAB会拒绝进行运算, 并会返回一个错误信息.

B.7 动 画

微分方程领域需要对动态系统或者说“动态的东西”进行研究. MATLAB让动画的制作变得简单, 而相关内容已在第 6 章详细讨论, 它们描绘一个随时间变化的微分方程的解.

MATLAB中的一个样例程序 **bounce.m** (见下面的程序代码) 展示了一个网球在单位方块的四壁之间的弹跳. 第一个 **set** 命令针对当前图像 (**gca**) 进行参数设置, 包括坐标轴的限制 $0 \leqslant x, y \leqslant 1$. **cla** 命令清除图像窗口, 而 **axis square** 命令把 x, y 方向的向量都变成单位长度.

接着, **line** 命令用于定义一条直线对象, 称为 **ball**, 并定义了 **ball** 的一些属性. **erase** 参数设置为 **xor**, 意思是每次描绘出网球的时候, 先前位置上的图像要擦掉. 在 **while** 循环之中的 4 个 **if** 命令使得小球碰到四面墙的时候有一个相反的速度. 循环也包含了一个 **set** 命令, 这个命令分别设置了 **xdata** 和 **ydata** 属性的值, 从而更新了直线对象 **ball** 当前的 x 轴坐标和 y 轴坐标. **drawnow** 命令在当前绘图窗口描绘出所有定义的对象. 小球运行的速度可以用 **pause** 命令来调整, 调整的步长从 **hx0** 到 **hy0**. **while** 循环是无限的, 可以用 ⟨Ctrl⟩+C 来终止. 下面是完整的程序:

```
%bounce.m
% Illustrates Matlab animation using the drawnow command
% Usage: Save this file in bounce.m, then type "bounce"
set(gca,'XLim',[0 1],'YLim',[0 1],'Drawmode','fast', ...
     'Visible','on');
cla
axis square
ball = line('color','r','Marker','o','MarkerSize',10, ...
       'LineWidth',2,'erase','xor','xdata',[],'ydata',[]);
hx0=.005;hy0=.0039;hx=hx0;hy=hy0;
xl=.02;xr=.98;yb=xl;yt=xr;x=.1;y=.1;
while 1 == 1
    if x < xl
        hx= hx0;
    end
    if x > xr
        hx = -hx0;
    end
    if y < yb
        hy = hy0;
    end
    if y > yt
        hy = -hy0;
    end
    x=x+hx;y=y+hy;
    set(ball,'xdata',x,'ydata',y);drawnow;pause(0.01)
end
```

参考文献

第 0 章

[1] D. Goldberg. What Every Computer Scientist Should Know about Floating Point Arithmetic. *ACM Computing Surveys* **23**, 5–48 (1991).

[2] N. J. Higham, *Accuracy and Stability of Numerical Algorithms.* SIAM Publishing, Philadelphia (1996).

[3] IEEE Standard for Binary Floating-Point Arithmetic, IEEE Std. 754–1985, IEEE, New York (1985).

[4] D. Knuth, *The Art of Computer Programming.* Addison-Wesley, Reading, MA (1981).

[5] M. Overton, *Numerical Computing with IEEE Floating Point Arithmetic.* SIAM Publishing, Philadelphia (2001).

[6] W. Stallings, *Computer Organization and Architecture,* 6th ed. Prentice Hall, Upper Saddle River, NJ (2003).

[7] J. Wilkinson, *Rounding Errors in Algebraic Processes.* Dover, New York (1994).

第 1 章

[1] R. P. Brent, *Algorithms for Minimization without Derivatives.* Prentice Hall, Englewood Cliffs, NJ (1973).

[2] C. G. Broyden, J. E. Dennis, Jr., J. J. Moré, "On the Local and Superlinear Convergence of Quasi-Newton's Methods." *IMA J. Applied Nath.* **12**, 223-245 (1973).

[3] A. S. Householder, *The Numerical Treatment of a Single Nonlinear Equation.* McGraw-Hill, New York (1970).

[4] J.-P. Merler, *Parallel Robots.* Kluwer Academic Publishers, London (2000).

[5] A. M. Ostrowski, *Solution of Equations and Systems of Equations*, 2d ed. Academic Press, New York (1966).

[6] J. F. Traub, *Iterative Methods for the Solution of Equations.* Prentice Hall, Englewood Cliffs, NJ (1964).

[7] J. Wilkinson, *Rounding Errors in Algebraic Processes.* Dover, New York (1994).

[8] J. Wilkinson, The Perfidious Polynomial. In: *Studies in Numerical Analysis*, Ed: G. Golub. MAA, Washington DC (1984).

第 2 章

[1] O. Axelsson, *Iterative Solution Methods.* Cambridge Univ. Press, New York (1994).

[2] C. G. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations." *Mathematics of Computation*, **19**, 577-593 (1965).

[3] J. W. Demmel, *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia (1997).

[4] J. E. Dennis, Jr., R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, Englewood Cliffs, New Jersey (1983).

[5] G. H. Golub, C. F. Van Loan, *Matrix Computations.* 2d ed. Johns Hopkins University Press, Baltimore (1989).

[6] W. Hackbush, *Iterative Solution of Large Sparse Systems of Equations.* Springer-Verlag, New York (1994).

[7] M. R. Hestenes, E. Steifel, "Conjugate Gradient Methods in Optimization." *J. Research National Bureau of Standards* **49**, 409–436 (1952).

[8] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Problems.* SIAM Publications, Philadelphia (1995).

[9] E. Anderson, Z. Bai, C. Bischof, J Demmel. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK: A Portable Linear Algebra Library for High-Performance Computers,* University of Tennessee, Knoxville, TN, CS-90-105, (1990).

[10] J. M. Ortega, *Numerical Analysis: a Second Course.* Academic Press, New York (1972).

[11] Y. Saad, *Iterative Methods for Sparse Linear Systems.* PWS-Kent Publishing, Boston (1996).

[12] G. W. Stewart, *Introduction to Matrix Computations.* Academic Press, New York (1973).

[13] J. Traub, *Iterative Methods for the Solution of Equations.* Prentice Hall, Englewood Cliffs, NJ (1964).

[14] N, Trefethen, D. Bau, *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics (1997).

[15] R. S. Varga, *Matrix Iterative Analysis*, 2d ed. Springer-Verlag, New York (2000).

[16] D. M. Young, *Iterative Solution of Large Linear Systems.* Academic Press, New York (1971).

第 3 章

[1] A. Davies, P. Samuels, *An Introduction to Computational Geometry for Curves and Surfaces.* Clarendon Press (1996).

[2] P. J. Davis, *Interpolation and Approximation.* Dover, New York (1975).

[3] C. de Boor, *A Practical Guide to Splines*, 2d ed. Springer-Verlag, New York (1984).

[4] G. Farin, *Curves and Surfaces for Computer-aided Geometric Design,* 2d ed. Academic Press, New York (1990).

[5] T. J. Rivlin, *An Introduction to the Approximation of Functions*, 2d ed. Dover, New York (1981).

[6] T. J. Rivlin, *Chebyshev Polynomials*, 2d ed. J. Wiley and Sons, New York (1990).

[7] M. H. Schultz, *Spline Analysis.* Prentice Hall Englewood Cliffs, NJ (1973).

[8] L. L. Schumaker, *Spline Function: Basic Theory.* Wiley, New York (1981).

[9] J. Volder, The CORDIC Trigonometric Computing Technique. *IRE Trans. Elec. Computing* **8**, 330-4(1959).

[10] F. Yamaguchi, *Curves and Surfaces in Computer-aided Geometric Design.* Springer-Verlag, New York (1988).

第 4 章

[1] N. Draper, H. Smith, *Applied Regression Analysis,* 3d ed. John Wiley and Sons, New York (2001).

[2] J. Fox, *Applied Regression Analysis, Linear Models, and Related Methods.* Sage Publishing (1997).

[3] G. Golub, C. Van Loan, *Matrix Computations*, 2d ed. Johns Hopkins University Press, Baltimore (1989).

[4] B. Hofmann-Wellenhof, H. Lichtenegger, J. Collins, *Global Positioning System: Theory and Practice,* 5th ed. Springer-Verlag New York (2001).

[5] C. L. Lawson, R. J. Hanson, *Solving Least Squares Problems.* SIAM Publications, Philadelphia (1995).

[6] T. Ryan, *Modern Regression Methods.* John Wiley and Sons (1997).

[7] G. Strang, K. Borre, *Linear Algebra, Geodesy, and GPS.* Wellesley Cambridge Press, Cambridge, MA (1997).

第 5 章

[1] H. Wang, J. Kearney, K. Atkinson, "Arc-length Parameterized Spline Curves for Real-time Simulation." In: *Curve and Surface Design:* Saint Malo 2002, Eds. T. Lyche, M. Mazure, L. Schumaker. Nashboro Press, Brentwood, TN (2003).

[2] P. Davis, P. Rabinowitz, *Methods of Numerical Integration*, 2d ed. Academic Press, New York (1984).

[3] H. Engels, *Numerical Quadrature and Cubature.* Academic Press, New York (1980).

[4] G. Evans, *Practical Numerical Integration.* J. Wiley and Sons, New York (1993).

[5] B. Guenter, R. Parent, "Motion Control: Computing the Arc Length of Parametric Curves." *IEEE Computer Graphics and Applications*, **10**, 72–78 (1990).

[6] S. Haber, "Numerical Evaluation of Multiple Integrals." *SIAM Review,* **12**, 481-526 (1970).

[7] A. Krommer, C. Ueberhuber, *Computational Integration.* SIAM Publishing, Philadelphia (1998).

[8] R. Piessens, E. de Doncker-Kapenga, C. Ueberhuber, D. Kahaner, *QUADPACK: A subroutine Package for Automatic Integration,* Springer, New York, NY (1983).

[9] A. Stroud, D. Secrest, *Gaussian Quadrature Formulas,* Prentice Hall, Englewood Cliffs, NJ (1966).

第 6 章

[1] U. M. Ascher and L. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-algebraic Equations.* Philadelphia: SIAM (1998).

[2] R. Ashino, M. Nagase and R. Vaillancourt, *Behind and Beyond the* MATLAB *ODE Suite.* CRM-2651, Jan. 2000 (and references therein).

[3] G. Birkhoff and G. Rota, *Ordinary Differential Equations,* 4th ed. J. Wiley & Sons (1989). NY, NY.

[4] P. Blanchard, R. Devaney, and G. R. Hall, *Differential Equations,* 2d ed. Brooks-Cole (2002). Pacitic Grove, CA.

[5] W. E. Boyce, and R. C. DiPrima, *Elementary Differential Equations and Boundary Value Problems*, 7th ed. J. Wiley & Sons (2003), NY, NY.

[6] M. Braun, *Differential Equations and Their Applications,* 4th ed. New York: Springer-Verlag (1993).

[7] J. C. Butcher, *Numerical Analysis of Ordinary Differential Equations.* London: Wiley (1987).

[8] *CODEE, ODE Architect Companion.* J. Wiley & Sons (1999). New York, NY.

[9] J. R. Dormand, *Numerical Methods for Differential Equations.* Boca Raton, FL: CRC Press (1996).

[10] C. Edwards and D. Penny, *Differential Equations and Boundary Value Problems,* 5th ed. Upper Saddle River, NJ: Prentice Hall (2004).

[11] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* Englewood Cliffs, NJ: Prentice Hall (1971).

[12] E. Harier, S. P. Norsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems,* 2d ed., Berlin: Springer-Verlag (1993).

[13] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems*, 2d ed., Berlin: Springer-Verlag (1996).

[14] P. Henrici, *Discrete Variable Methods in Ordinary Differential Equations.* New York, J. Wiley & Sons NY (1962).

[15] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations,* Cambridge, UK Cambridge University Press (1996).

[16] E. Kostelich, D. Armbruster, *Introductory Differential Equations: From Linearity to Chaos.* Boston MA Addison Wesley (1997).

[17] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems,* Chichester: John Wiley & Sons (1991).

[18] P. J. McKenna, C. Tuama, "Large Torsional Oscillations in Suspension Bridges Visited Again: Vertical Forcing Creates Torsional Response." *Amer. Math. Monthly* **108**, 738–754 (2001).

[19] J. Polking, *Ordinary Differential Equations Using MATLAB.* Upper Saddle River, NJ: Prentice Hall (1999).

[20] L. F. Shampine, *Numerical Solution of Ordinary Differential Equations.* New York: Chapman & Hall (1994).

[21] L. F. Shampine, I. Gladwell, and S. Thompson. *Solving ODEs with MATLAB.* Cambridge University Press (2003).

[22] L. F. Shampine and M. W. Reichelt. "The MATLAB ODE Suite." *SIAM J. Sci. Computing* **18**, 1–22 (1997).

[23] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis,* 2d ed. New York: Springer-Verlag (1993).

第 7 章

[1] U. M. Ascher, R. M. Mattheij, R. B. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations.* SIAM, Philadelphia (1995).

[2] S. Brenner, L. R. Scott, *The Mathematical Theory of Finite Element Methods.* Springer Verlag, New York (1994).

[3] C. S. Desai, T. Kundu, *Introductory Finite Element Method.* CRC Press, Boca Raton, FL (2001).

[4] J. V. Huddleston, *Extensibility and Compressibility in One-dimensional Structures,* 2d ed. ECS Publishing, Buffalo (2000).

[5] H. B. Keller, *Numerical Methods of Two-Point Boundary-Value Problems.* Blaisdell, Waltham, MA (1968).

[6] P. B. Bailey, L. F. Shampine, P. E. Waltman, *Nonlinear Two-Point Boundary-Value Problems.* Academic Press, New York (1968).

[7] S. Roberts, J. Shipman, *Two-Point Boundary Value Problems: Shooting Methods.* Elsevier, New York (1972).

第 8 章

[1] W. F. Ames, *Numerical Methods for Partial Differential Equations,* 3d ed. Academic Press, Boston (1992).

[2] R. Bank, *PLTMG, A Software Package for Solving Elliptic Partial Differential Equations*, Users'Guide 8.0. SIAM Publications, Philadelphia (1998).

[3] S. Brenner, L. R. Scott, *The Mathematical Theory of Finite Element Methods.* Springer Verlag, New York (1994).

[4] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems.* North-Holland, Amsterdam (1978).

[5] R. Courant, K. O. Friedrichs, H. Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik." *Math. Annalen* **100**, 32–74 (1928).

[6] H. Elman, D. J. Silvester, A. Wathen, *Finite Elements and Fast Iterative Solvers.* Oxford University Press, Oxford, U. K. (2004).

[7] L. C. Evans, *Partial Differential Equations,* AMS Publications, Providence, RI (2002).

[8] M. Gockenbach, *Partial Differential Equations: Analytical and Numerical Methods.* SIAM, Philadelphia (2002).

[9] R. Haberman, *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems.* Prentice Hall, Upper Saddle River, NJ (2004).

[10] C. Hall, T. Porsching, *Numerical Analysis of Partial Differential Equations.* Prentice Hall, Englewood Cliffs, NJ (1990).

[11] L. Lapidus, G. F. Pinder, *Numerical Solution of Partial Differential Equations in Science and Engineering.* Wiley-Interscience, New York (1982).

[12] J. D. Logan, *Applied Partial Differential Equations.* Wiley, New York (1994).

[13] A. R. Mitchell, D. F. Griffiths, *The Finite Difference Method in Partial Differential Equations.* Wiley, New York (1980).

[14] K. W. Morton, D. F. Mayers, *Numerical Solution of Partial Differential Equations,* Cambridge University Press, Cambridge, U. K. (1996).

[15] J. R. Rice and R. F. Boisvert, *Solving Elliptic Problems Using ELLPACK* Springer-Verlag, New York (1984).

[16] G. Strang, G. J. Fix, *An Analysis of the Finite Element Method.* Prentice Hall, Englewood Cliffs, NJ (1973).

[17] J. N. Strikwerda, *Finite Difference Schemes and Partial Differential Equations.* Wadsworth and Brooks-Cole, Pacific Grove, CA (1989).

[18] W. A. Strauss, *Patrial Differential Equations: An Introduction.* J. Wiley and Sons, New York (1992).

第 9 章

[1] F. Black, M. Scholes, "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy,* **81**, 637–654 (1973)

[2] G. E. P. Box, M. Muller, "A Note on the Generation of Random Normal Deviates." *Ann. Math. Stat.* **29**, 610-611 (1958).

[3] G. S. Fishman, *Monte Carlo: Concepts, Algorithms*, and Applications. Springer-Verlag, New York (1996).

[4] J. E. Gentle, *Random Number Generation and Monte Carlo Methods,* 2d ed. Springer-Verlag, New York (2003).

[5] J. H. Halton, "On the Efficiency of Certain Quasi-Random Sequences of Points in Evaluating multi-Dimensional Integrals" *Numerische Mathematik* 2, 84–90 (1960).

[6] P. Hellekalek, "Good Random Number Generators Are (Not So) Easy to Find" *Mathematics and Computers in Simulation*, **46**, 485-505 (1998).

[7] D. J. Higham, "An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations." *SIAM Review* **43**, 525–546 (2001).

[8] J. C. Hull, *Options, Futures, and Other Derivatives,* 5th ed. Prentice Hall, Upper Saddle River, NJ (2002).

[9] F. Klebaner, *Introduction to Stochastic Calculus with Applications.* Imperial College Press, London (1998).

[10] P. Kloeden, E. Platen, *Numerical Solution of Stochastic Differential Equations.* Springer-Verlag, Berlin (1992).

[11] P. Kloeden, E. Platen, H. Schurz, *Numerical Solution of SDE through Computer Experiments.* Springer-Verlag, Berlin (1994).

[12] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms,* 3d ed. Addison-Wesley. Reading, MA (1997).

[13] G. Marsaglia, "Random Numbers Fall Mainly in the Planes." *Proc. Acad. Sci.* **61**, 25 (1968).

[14] G. Marsaglia, A. Zaman, "A New Class of Random Number Generators." *Annals of Applied Probability* **1**, 462–480 (1991).

[15] G. Marsaglia and W. W. Tsang, "The Ziggurat Method for Generating Random Variables," *Journal of Statistical Software* **5**, 1–7 (2000).

[16] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM Publications, Philadelphia (1992).

[17] B. Oksendal, *Stochastic Differential Equations: An Introduction with Applications*, 5th ed. Springer-Verlag, Berlin (1998).

[18] S. Park, K, Miller, "Random Number Generators: Good Ones Are Hard to Find." *Communications of the ACM* 31, 1192–1201 (1988).

[19] R. Y. Rubinstein, *Simulation and the Monte Carlo Method.* J. Wiley, New York (1981).

[20] J. M. Steele, *Stochastic Calculus and Financial Applications.* Springer-Verlag, New York (2001).

[21] P. Wilmott, S. Howison, J. Dewynne, *The Mathematics of Financial Derivatives.* Cambridge University Press, Oxford and New York (1995).

第 10 章

[1] R. Bracewell, *The Fourier Transform and Its Application*, 3d ed. McGraw-Hill, New York (2000).

[2] W. Briggs and V. E. Henson, *The DFT: An Owner's Manual for the Discrete Fourier Transform.* SIAM Publications, Philadelphia (1995).

[3] E. O. Brigham, *The Fast Fourier Transform and Its Applications.* Prentice Hall Englewood Cliffs, NJ(1988).

[4] E. Chu and A. George, *Inside the FFT Black Box.* CRC Press, Boca Raton, FL (1999).

[5] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series." *Math. Comp.* 19, 297–301 (1965).

[6] P. Duhamel and M. Vetterli, "Fast Fourier Transforms: A Tutorial Review and a State of the Art." *Signal Processing* **19**, 259–299 (1990).

[7] M. Frigo and S. G. Johnson, " FFTW: An Adaptive Software Architecture for the FFT." *Proceedings ICASSP* **3**, 1381–1384 (1998).

[8] P. N. Swarztrauber, "Vectorizing the FFTs." In: *Parallel Computations,* ed. G. Rodrigue, pp. 51–83. Academic Press, New York (1982).

[9] A. Oppenheim, R. Schafer and J. Buck, *Discrete-time Signal Processing*, 2d ed. Prentice Hall, Upper Saddle River, NJ (1999).

[10] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform.* SIAM Publications, Philadelphia (1992).

[11] S. Winograd, "On Computing the Discrete Fourier Transform." *Math. Comp.* **32**, 175–199 (1978).

第 11 章

[1] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures.* Kluwer Academic Publishers, Boston (1995).

[2] K. Brandenburg and M. Bosi, "Overview of MPEG Audio: Current and Future Standards for Low Bit Rate Audio Coding." *J. Audio Eng. Soc.* **45**, 4–21 (1997).

[3] S. Hacker, *MP3: The Definitive Guide.* O'Reilly Publishing, Sebastopol, CA (2000).

[4] D. A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes." *Proceedings of the IRE*, 40, 1098–1101 (1952).

[5] H. S. Malvar, *Signal Processing with Lapped Transforms.* Artech House, Norwood, MA (1992).

[6] M. Nelson and J. Gailly, *The Data Compression Book*, 2d ed. M & T Books, Redwood City, CA (1995).

[7] W. Pennebaker and J. Mitchell, *JPEG Still Image Data Compression Standard.* Van Nostrand Reinhold, New York (1993).

[8] K. R. Rao and J. J. Hwang, *Techniques and Standards for Image, Video, and Audio Coding.* Prentice Hall, Upper Saddle River, NJ (1996).

[9] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications.* Academic Press, Boston (1990).

[10] K. Sayood, *Introduction to Data Compression.* Morgan Kaufmann Publishers, San Francisco (1996).

[11] J. A. Storer, *Data Compression: Methods and Theory.* Computer Science Press, Rockville, MD (1988).

[12] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice.* Kluwer, Boston (2002).

[13] G. K. Wallace, "The JPEG Still Picture Compression Standard." *Communications of the ACM*, **34**, 30–44 (1991).

[14] Y. Wang and M. Vilermo, "The Modified Discrete Cosine Transform: Its Implications for Audio Coding and Error Concealment." *J. Audio Eng. Soc.* **51**, 52–62 (2003).

第 12 章

[1] E. Anderson, Z. Bai, C. Bischof, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, "LAPACK: A Portable Linear Algebra Library for High-performance Computers," *Computer Science Dept. Technical Report* CS-90-105, University of Tennessee, Knoxville (1990).

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. Van der Vorst, eds. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* SIAM, Philadelphia (2000).

[3] S. Brin and L. Page, "The Anatomy of a Large-scale Hypertextual Web Search Engine." In: *Proc. 7th Int. World Wide Web Conference* (1998).

[4] J. Cuppen, "A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem," *Numerische Math.* **36**, 177-195 (1981).

[5] G. Golub and C. Van Loan, *Matrix Computations,* 2d ed. Johns Hopkins University Press, Baltimore (1989).

[6] B. Parlett, *The Symmetric Eigenvalue Problem.* SIAM, Philadelphia (1998).

[7] B. Parlett, "The QR Algorithm." *Computing in Sci. and Eng.* **2**, 38–42 (2000).

[8] G. Pinski and F. Narin, "Citation Influence for Journal Aggregates of Scientific Publications: Theory, with Application to the Literature of Physics". *Information Processing and Management* **12**, 297–312 (1976)

[9] Y. Saad, *Numerical Methods for Large Eigenvalue Problems.* Halstead Press, New York (1992).

[10] B. T. Smith, J. M. Boyle, Y. Ikebe, V. Klema, and C. B. Moler, *Matrix Eigensystem Routines: EISPACK Guide,* 2d ed. Springer-Verlag, New York (1970).

[11] G. W. Stewart, *Introduction to Matrix Computations.* Academic Press, New York (1973).

[12] D. S. Watkins, "Understanding the QR Algorithm." *SIAM Review* **24**, 427–440 (1982).

[13] J. Wilkinson, *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford (1965).

[14] J. Wilkinson and C. Reinsch, *Handbook for Automatic Computation, Vol. 2: Linear Algebra.* Springer-Verlag, New York (1971)

第 13 章

[1] J. Dennis and Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, Englewood Cliffs; NJ (1983).

[2] C. A. Floudas, P. M. Pardalos, C. Adjiman, W. R. Esposito, Z. H. Gms, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger, *Handbook of Test Problems in Local and Global Optimization*, Vol. 33, Nonconvex Optimization and its Applications. Springer, Berlin (1999).

[3] J. C. Lagarias, J. A. Reeds, M. H. Wright and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions." SIAM *Journal of Optimization,* 9, 112-147, 1998.

[4] J. Moré and S. Wright, *Optimization Software Guide.* SIAM, Philadelphia.

[5] S. Nash and A. Sofer, *Linear and Nonlinear Programming,*. McGraw-Hill, New York (1996).

[6] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization." *Computer J.* **7**, 308–313 (1965)

[7] J. Nocedal and S. Wright, *Numerical Optimization*, Springer Series in Operations Research. Springer, New York (1999).

［Ｇｅｎｅｒａｌ　Ｉｎｆｏｒｍａｔｉｏｎ］
书名＝数值分析
ＳＳ号＝１２３９４８８３